# e-Commerce Vostok

**v1.1.4**

## Content

**Overview** 4
**Step-by-step integration process** 6
**1. Payment process** 7
**1.1. Payment process flow for Hosted Checkout integration** 7
1.1.1. Regular payment. 7
1.1.2. 2-step payment. 7
1.1.3. Process description 8
**1.2. Payment process flow for direct integration with e-Commerce API** 10
1.2.1. Process description 10
**1.3. Card tokenization** 11
**1.4. Details of the partner and merchant** 13
**2. e-Com API** 13
**2.1. Requests Encryption** 14
2.1.1. Key generation 14
2.1.2. Transfer the public key to the Bank 14
2.1.3. Signature 15
2.1.4. Creating a private key hash and sending X-Signature and X-Key 15
2.1.5. Signature example 15
**2.2. eCom SDK** 16
2.2.1. eCom SDK fields description 17
2.2.2. eCom Javascript SDK 18
2.2.3. API e-Com SDK 19
**2.3. eCom API methods** 20
2.3.1. /api/order/getOrder 20
2.3.2. /api/order/pay 21
2.3.3. /api/order/cancel 22
2.3.4. /payInfo (callback) 24
2.3.5. /api/order/create 24
2.3.6. /api/order/approve 26
2.3.7. /api/order/confirm 32
2.3.8. /api/cardToken/getUrl 33
2.3.9. /api/cardToken/getAll 35
2.3.10. /api/cardToken/updateAlias 37
2.3.11. /api/cardToken/delete 37
**2.4. Google Pay** 39
2.4.1. e-Commerce Hosted Checkout integration 39
2.4.2. Direct integration with Google Pay API 39
**2.5. Apple Pay** 40

| | |
|---|---|
| 2.5.1. e-Commerce Hosted Checkout integration | 40 |
| 2.5.2. Direct integration with Apple Pay API | 40 |
| **2.6. Order statuses** | **42** |

## Overview

**E-commerce Vostok** – a payment platform that provides capabilities for merchants to pay various purchases using a range of accepted payment methods. PCIDSS certification and 3D Secure 2.0 support allow you to process transactions fast and securely on the payment page.

All available payment methods are displayed to the client after redirecting to the checkout page. The list of the methods are dependent on the operating system, browser, and order details. More details about each of them are given below.

### Available payment methods

#### Payment by card

The capability to pay for the order by entering the card details – number, expiration date, and CVC2 code.  Cards of MasterCard, Visa, and Prostir payment systems are accepted for transactions. If the card supports 3D Secure, the client will be redirected to the bank-issuer page to confirm the transaction.

There are no limitations on the client's operating system or browser while using this payment option.

#### Google Pay™

Google Pay™ – is a quick and easy way that allows you to make payments by card details without entering them for each payment. Your card details don't appear on your payments page and are securely stored in Google. Irrespective of the operating system or web browser, this payment mechanism is compatible with both mobile and web-based devices.

For cards that are added to the GPay wallet directly (without tokenization), the client may be redirected to the 3D Secure check page according to the rules similar to when paying by entering the card details.

By using GPay payment method you agree with Google Pay APIs Acceptable Use Policy and accept the terms defined in the Google Pay API Terms of Service.

There are 2 ways for integration with the GPay payment method - e-Commerce Hosted Checkout or direct integration with Google Pay API. More details in the Google Pay chapter below.

**Please note!** Google Pay doesn’t work at e-Com Checkout in WebView. For mobile platforms it is recommended to open e-Com Checkout in the default browser or use direct integration with Google Pay API with further sending GPay token to e-Com direct paymentAPI.

## BVRPay

Fast payments for cardholders of “Банк Власний Рахунок”. In order to confirm the payment, the push is sent to the mobile application, and after confirmation, the funds are debited from the main account of the Bank's client account. Since the card details are not used on the payment page, additional 3D Secure checks are unnecessary.

This payment method is available in case of filling the client's mobile phone number in the order details. If a Bank's client is found using this phone number, a push confirmation will be sent to him. For the fraud-prevention, the fact of push sending is displayed as successful, regardless of whether the client is registered or not.

## Apple Pay™

Apple Pay™ – is a quick and easy way for owners of the iOS operating system and the Safari browser, which allows you to make payments using card details without entering them for each payment.

Apple Pay is one of the fastests payment methods because it doesn’t require additional 3DSecure verification.

There are 2 ways for integration with the Apple Pay payment method - e-Commerce Hosted Checkout or direct integration with Apple Pay API. More details in the Apple Pay chapter below.

## Step-by-step integration process

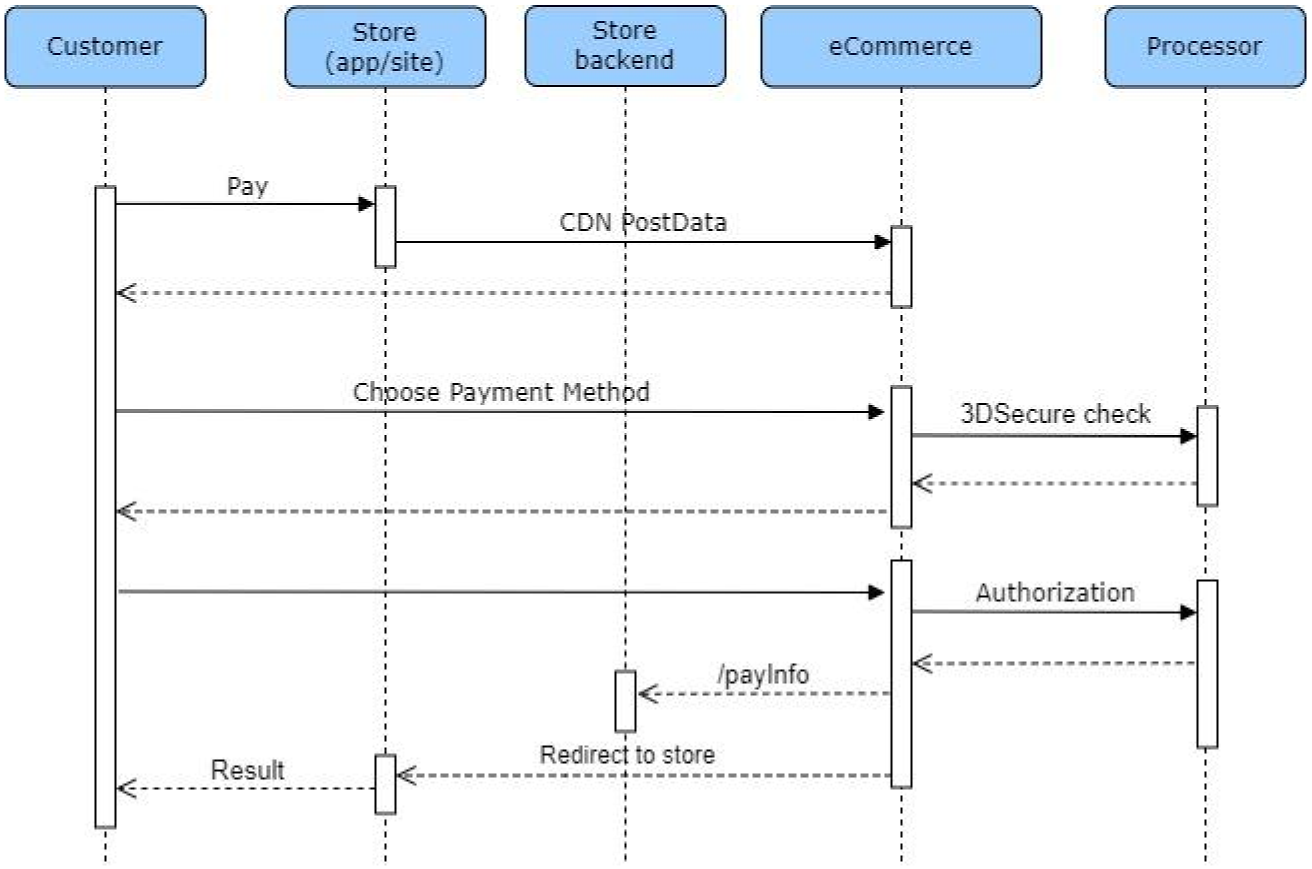
1. The partner reads current documentation.
2. Partner generates the private key for the test environment and provides to the Bank corresponding public key and details of the partner and merchant for setting up on the test environment.
3. Bank generates and provides test **merchantid**.
4. Partner integrates on the test environment:
    a. eCom SDK for Checkout redirection (in case of using e-Commerce Hosted Checkout).
    b. eCom API for sending requests from BackEnd of partner's store.
5. Partner executes test payments and provides to the Bank their unique identifiers (**partnerOrderId**) for approval.
6. After the Bank checks and approves test payments, the partner generates the private key for the production environment and provides to the Bank corresponding public key.
7. Bank generates and provides production **merchantid**.
8. Partner executes production payments and provides to the Bank their unique identifiers (**partnerOrderId**) for approval.
9. After the Bank checks and approves production payments, the partner is ready to activate integration to all store's payments.

## 1. Payment process

### 1.1. Payment process flow for Hosted Checkout integration

#### *1.1.1. Regular payment.*

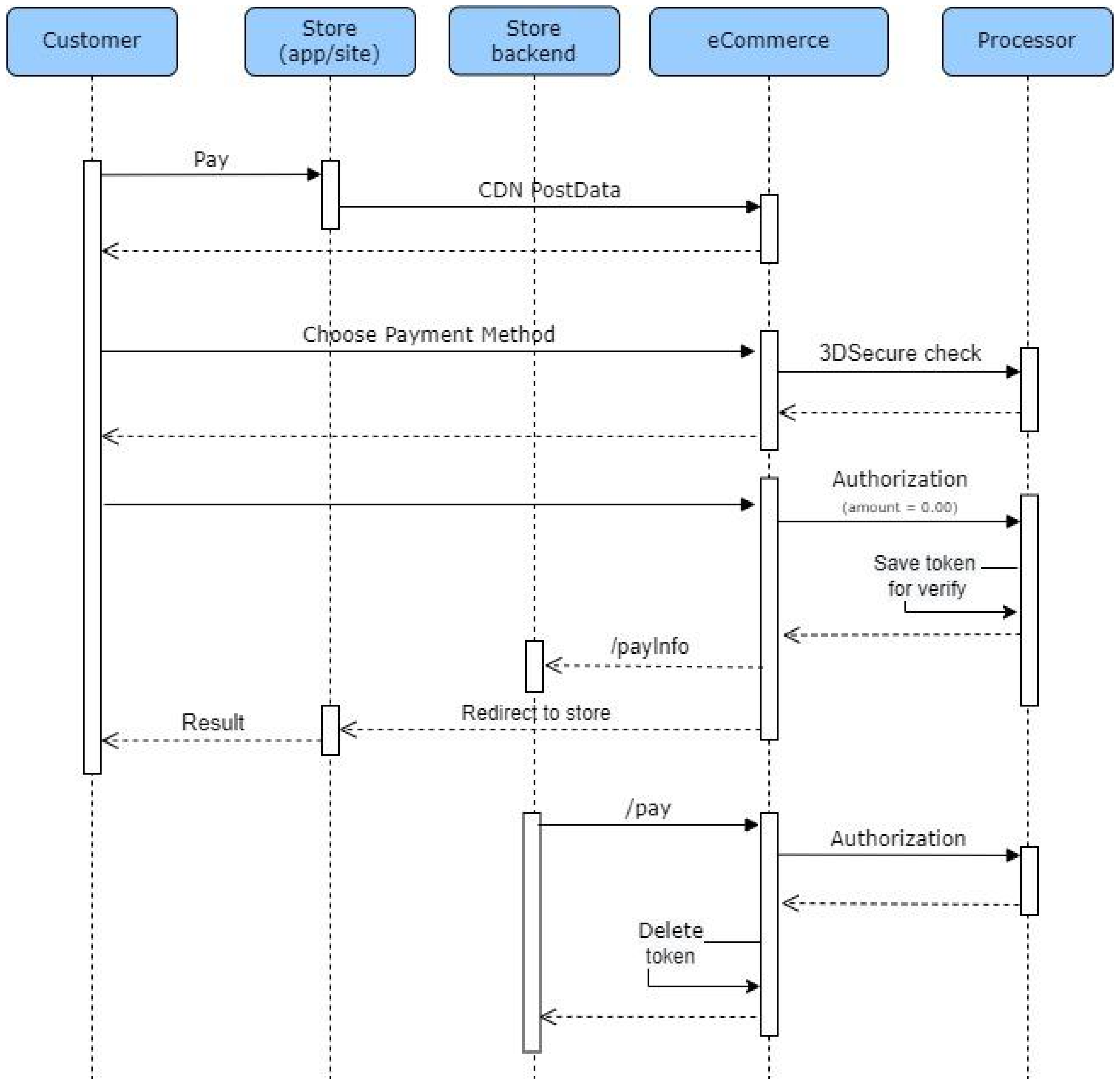
Debiting funds from the client's card during payment processing.



#### *1.1.2. 2-step payment.*

Debiting funds from the client's card at the time after completing the order in the Store:

- authType = PreAuthorization – for an amount which does not exceed the original amount of the order;
- authType = Verify – for any amount.

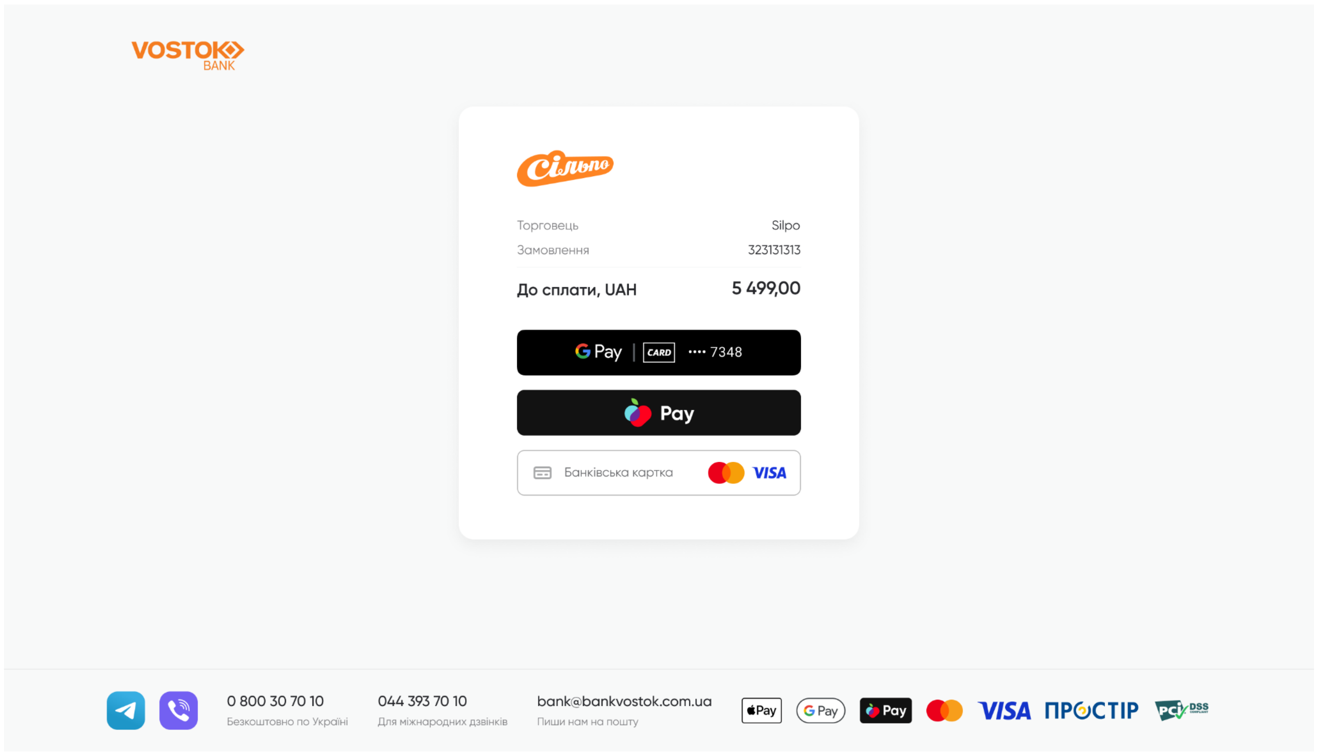


### 1.1.3. *Process description*

| **Regular payment**<br>**(authType = Authorization)** | **2-step payment**<br>**(authType = PreAuthorization, Verify)** |
|---|---|
| 1. The Customer fills the cart in the Store and clicks Pay. | |
| 2. The Store sends a request to e-commerce and redirects the client to eCom Checkout. | |
| 3. The customer chooses the appropriate payment method on Checkout (by card, GPay, APay, BVRPay) and confirms the payment. | |
| 4. E-commerce sends a request to the Processor in order to check if 3DSecure is enabled on the client's card.<br>4.1. If 3DSecure is enabled on card, the client will be forwarded to the issuing bank's page to confirm the transaction. | |

| | |
|---|---|
| 4.2. For the BVRPay payment method, instead of checking 3DSecure, the push confirmation in the “Банк Власний рахунок” application is used. | |
| 5. After successful verification of 3DSecure - eCom Checkout sends a request to the Processor to debit funds from the client's card. **The full amount of the order is debited.** | 6. After successful verification of 3DSecure – eCom Checkout sends a request to the Processor to debit funds from the client's card.<br>6.1. For authType = Verify, a 0 amount is debited (regardless of the amount transferred in the order details) with the generation of a one-time-use card token for a further finally formed order amount. The token is disposable and valid only for the current order.<br>6.2. For authType = PreAuthorization, eCom Checkout sends a request to the Processor to debit the amount transferred in the order details with the generation of a one-time-use card token for further post-authorization, which **cannot exceed the amount transferred in the PreAuthorization request.** |
| 7. The result of debiting funds is sent to the Store’s callbackUrl. | |
| 8. Checkout redirects the Customer to the corresponding Store page:<br>8.1. If the payment is successful, redirects to SuccessRedirectUrl.<br>8.2. If the payment results in an error, redirects to failureRedirectUrl. | |
| | 9. After the Store finalizes an order and determines its final amount, it must send a request to the eCom API (pay method) to finally pay for the order. |

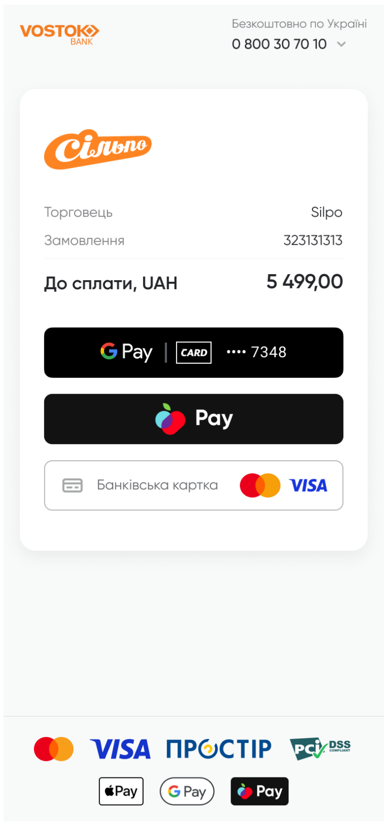
An example of displaying checkout for web:



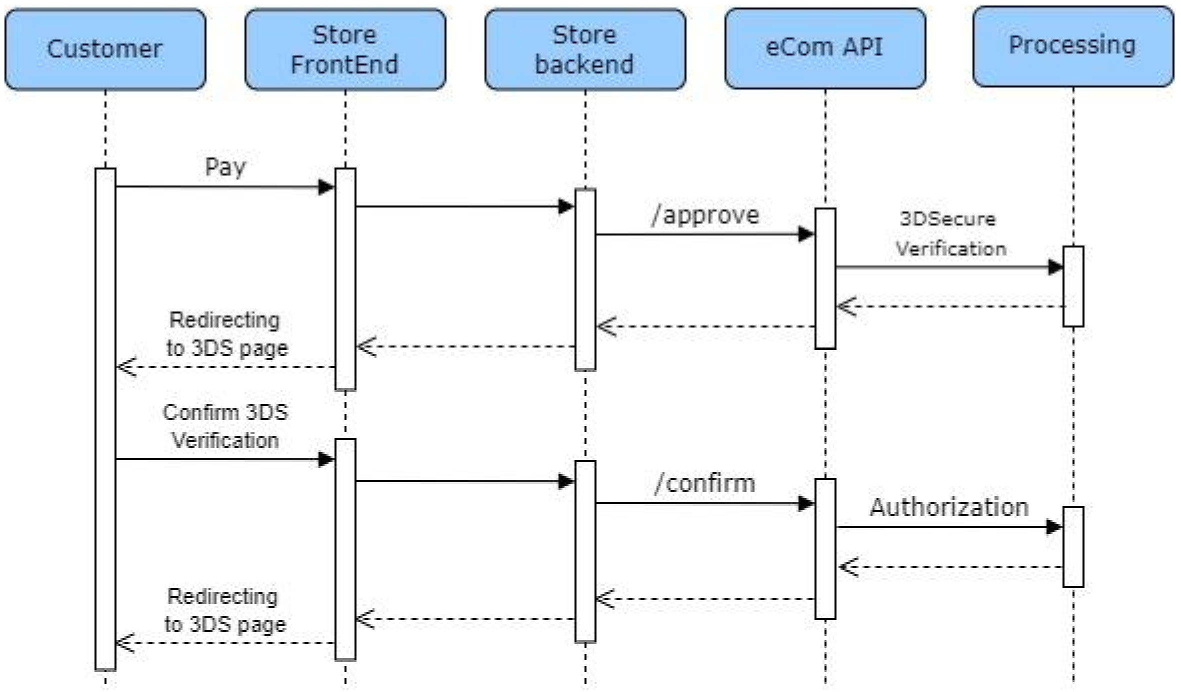
Торговець Silpo
Замовлення 323131313
До сплати, UAH 5 499,00

0 800 30 70 10 Безкоштовно по Україні
044 393 70 10 Для міжнародних дзвінків
bank@bankvostok.com.ua Пиши нам на пошту

An example of displaying checkout for mobile:



### *1.2.* Payment process flow for direct integration with e-Commerce API



#### *1.2.1. Process description*

1. The customer adds an item to the cart and proceeds to payment.

2. The store's backend calls register and check purchase method (approve).
3. e-Commerce checks request data and if availability of 3DSecure verification.
4. If 3DSecure verification needed - e-Commerce returns URL in the response to proceed the verification.
5. The store redirects the customer to 3DSecure verification URL.
6. Customer proceeds 3DSecure verification and returns to the store.
7. The store's backend calls confirm the purchase method (confirm).
8. e-Com authorizes the payment.
9. The store receives payment results in response.

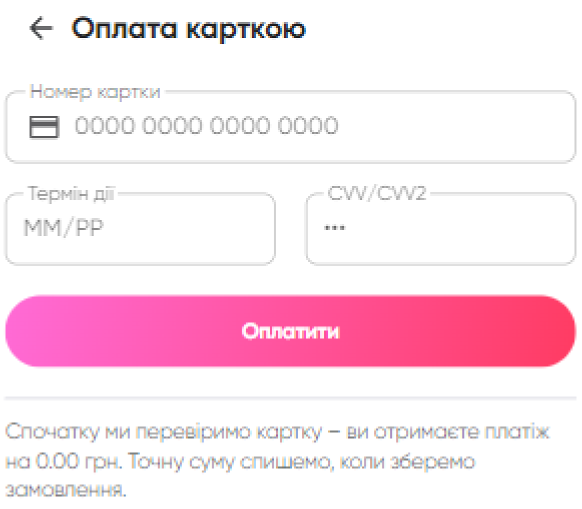
### *1.3.* Card tokenization

Card tokenization allows the partner to perform recurring payments of customer orders through the eCom API without their involvement.

There are 2 types of tokenization:

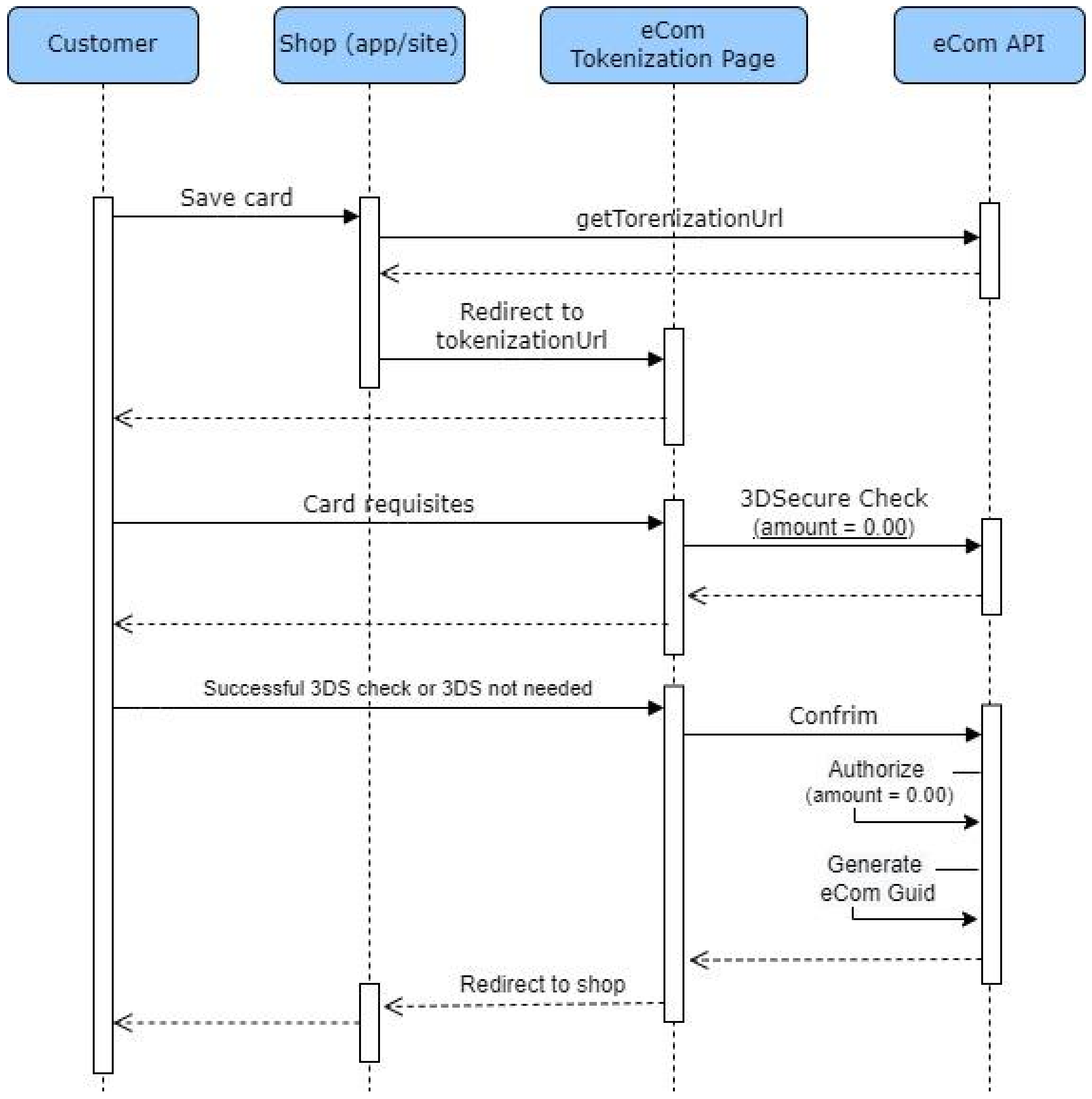
- **card tokenization for a specific order** - in this case, it is necessary to specify the filled parameter *partnerOrderId* in the getUrl request - the card will be tokenized for a single payment of this order (the token will be automatically utilized after a successful payment);
- **card tokenization for multiple orders** - in this case, it is necessary **not to specify and fill** the parameter *partnerOrderId* in the getUrl request - the card will be tokenized on a permanent basis and the token will be available for payment of **any partner's merchant** who performed the tokenization.

An example of displaying tokenization page:



Спочатку ми перевіримо картку – ви отримаєте платіж на 0.00 грн. Точну суму спишемо, коли збережемо замовлення.

The tokenization process is shown at the diagram:



1. The customer on the partner's website or app selects the option "Add card".
2. The partner sends a request to get a link for tokenization (getUrl).
3. eCom returns a generated link of tokenization page in response.
4. The partner's website (app) redirects the customer via this link.
5. The customer on the tokenization page fills the card details - number, expiration date and CVV.
6. eCom performs 3DS-verification of the card. If 3DS is needed - the customer will be redirected to the card issuer's bank page.
7. In case of successful 3DS-verification or if it is not required for the card - eCom performs authorization of debiting the amount of 0.00 UAH on the card.
8. eCom generates the card's tokenGuid and sends it to the partner's *callbackUrl*, which was specified when generating the link for tokenization in the getUrl method.
   a. If a customer opens tokenization URL after successful tokenization - eCom will redirect to successRedirecUrl automatically during 24 hours from the moment when tokenization URL

was created. After 24 hours customer tokenization URL will be utilized and the customer will receive an error.

9. In the future, the partner can use the received token for payment with *cardDataType = tokenGuid* in the approve method.

The following methods are used with existing tokens:

- getAll - getting all tokenized cards by the unique customer identifier in the partner's system;
- updateAlias - updating the name for the tokenized card;
- delete - deleting the tokenized card.

Each method is described in the corresponding chapter of the e-Com API.

### *1.4.* Details of the partner and merchant

**A partner** is an organization or individual entrepreneur with whom an agreement has been concluded.

**A merchant** is a partner's store (site) where orders will be paid. One partner can have an unlimited number of merchants.

To register a **partner** in the e-commerce, the following details are required:

- *Partner's name*

To register **a merchant** in the e-commerce, the following details are required:

- *Merchant's name - the* name of the store, which will be displayed on the Checkout.
- *Merchant logo* – file or URL with the merchant's logo, which will be displayed on the Checkout.

Additional Urls parameters:

- *defaultSuccessRedirectUrl* (type: *string*) – URL of the merchant's store by default, where the client will be redirected after successful payment.
- *defaultFailureRedirectUrl* (type: *string*) – URL of the merchant's store by default, where it will be redirected if the payment is completed with an error.
- *defaultDeepLinkUrl* (type: *string*) – DeepLink of the merchant's mobile application, through which the client will be redirected after confirming the payment to Checkout.

If the Store uses unique *successRedirectUrl*, *failureRedirectUrl*, and *deepLinkUrl* for each order then these parameters do not need to be filled.

After successful registration, a unique **partner ID** and **merchant ID** are generated for each partner and merchant. They are further used in requests to the eCom API.

## 2. e-Com API

### 2.1. Requests Encryption

A signature is used to validate requests.

When submitting requests, you must send the signature value in the «**signature**» field, which will be used to validate requests from a partner.

An asymmetric pair of keys (private-public keys) is formed for each partner.

Information exchange is carried out via mail in the Shared password archive.

To receive the Shared password, you must write an email request to s.lugovskyi@bankvostok.com.ua. A shared password will be provided in the response.

#### 2.1.1. *Key generation*

To successfully process requests, you need to generate a pair of keys using the OpenSSL utility:

1. Generate a private key:

```
openssl genrsa -out merchant.key 2048
```

2. Generate a public key. Obtaining a public key in PEM format:

```
openssl rsa -in merchant.key -pubout -out pubkey.pem
```

3. After generating a private key, you will get a hash of the private key (according to the HMAC256 algorithm).

**Creating a private key hash**

```
sha256Hash.ComputeHash(keyDataByteArray);
```

(!) Please note that the byte array of the generated private key must be set to the hash function. In the future, it will be used to check the *X-Signature Header*.

#### 2.1.2. *Transfer the public key to the Bank*

After generating the files of the pubkey.pem file, you need to send to the Bank:

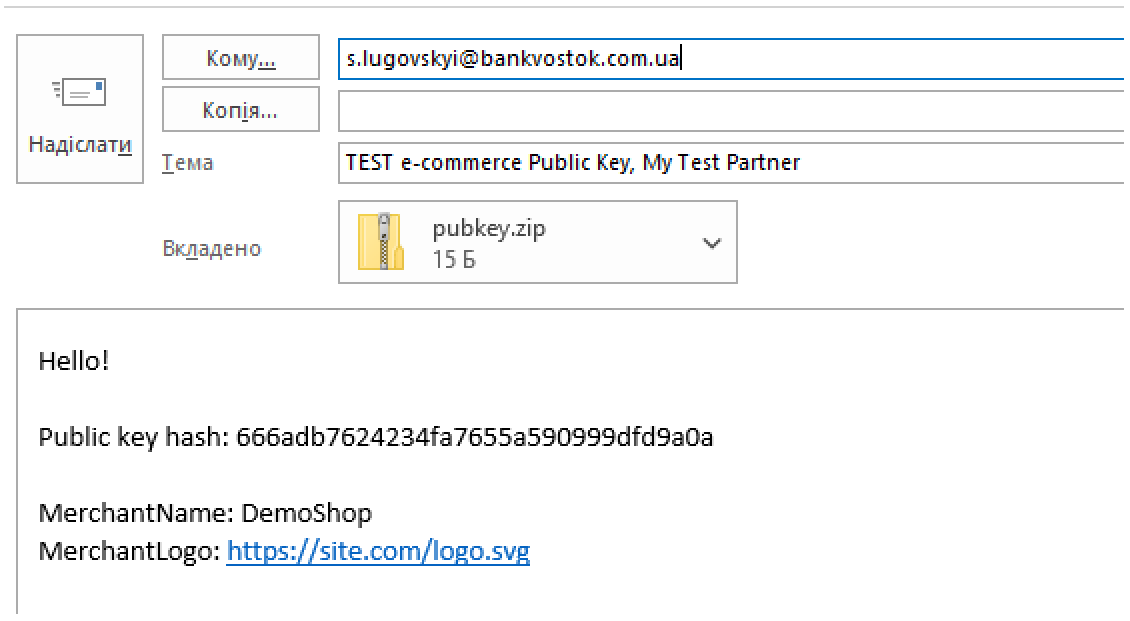
- Partner's public key *(pubkey.pem)*
- Private key hash (**NOT private key!**)

**Please note: Partner's private key must be saved and be accessible only to the partner. Please do not send your private key to anyone, including the Bank.** e-Commerce needs only **private key hash** and corresponding **public key** for signature check.

The data above must be transferred in the Shared Password archive to s.lugovskyi@bankvostok.com.ua.

*Email Header: {Environment} e-commerce Public Key, {Partner Name}.*

Email example:



s.lugovskyi@bankvostok.com.ua

TEST e-commerce Public Key, My Test Partner

pubkey.zip
15 Б

Hello!

Public key hash: 666adb7624234fa7655a590999dfd9a0a

MerchantName: DemoShop
MerchantLogo: https://site.com/logo.svg

After registering the public key with the Bank, the partner will receive a response that can be used to form a signature with this key.

#### 2.1.3. *Signature*

Signature is formed from the original request using the RSA algorithm with the client's private key.

The 2048-bit RSA key and the SHA-256 cryptographic algorithm must be used to create a signature.

To obtain a signature, use the following formula:

**`BASE64(RSA_SIGN(private_key, data, sha256 digest))`**

*`private_key`* – partner's private key generated with OpenSSL;

*`data`* – request header;

*digest* – cryptographic algorithm SHA-256.

Next, we apply the **RSA_SIGN()** method of public key encryption algorithm to a given data set.

To send it in the request, we use the **BASE64()** function to convert the resulting array to BASE64 format.

To convert a byte array to a string **Encoding.UTF-8** is used.

#### 2.1.4. *Creating a private key hash and sending X-Signature and X-Key*

To verify the request, you need to send this hash key in the request header.

**Header: X-Key**

**Value: SHA256HASH(private_key)**

**Header: X-Signature**

**Value: BASE64(RSA_SIGN(private_key, data, sha256 digest))**

This is necessary to authenticate the request from the client.

#### 2.1.5. *Signature example*

*C# example*

a) We form a pem-certificate and download it for signature formation.

**Downloading the certificate:**

```
var keyString = @"-----BEGIN RSA PRIVATE
KEY-----SDFSAGFGFDFGSDGREJIGVFIJFVISJFSDIMSDIGVFDSIJFGIFSSDFIJGDS-----END RSA PRIVATE KEY-----";
var rsa = RSA.Create();
rsa.ImportFromPem(keyString.ToCharArray());
```

b) Data generation

**Checkout** – on the base fields of the query and their values.

For the data string formation special pipe symbol is used: "|"

A row is generated automatically in a hidden form field *signatureValidationString* in the following format:

`"{partnerOrderId}|{amount}|{merchantId}|{authType}"` – all values of the submitted query fields.

`{partnerOrderId}` – merchant's unique order identifier, generated on the merchant's side.

`{amount}` – the order amount is in numerical format, the delimiter is a period, with the required 2 characters after the period. Examples: 0.05, 10.00, 30.10, 47465.25;

`{merchantId}` – merchant's unique identifier, which is provided during the registration in e-commerce.

`{authType}` – in numeric (1 – Authorization, 2 – PreAuthorization, 4 – Verify).

The order of the fields must correspond to the order of the fields indicated above.

**API** – data is the original request that the partner sends to the e-commerce API (request body JSON in minified format - without tabs, spaces and line breaks).

c) After that, we generate a signature using the generated data for the request:

```
// C# example code, generation signature string based on parameters
var signatureValidationString = "order1234|234.00|123456|1";
var byteString = Encoding.UTF8.GetBytes(signatureValidationString);
var signature = rsa.SignData(byteString, HashAlgorithmName.SHA256, RSASignaturePadding.Pkcs1);
return Convert.ToBase64String(signature);
```

## 2.2. eCom SDK

There are 2 ways for integration with eCom SDK and redirection to Checkout:

1. **JS eCom SDK library.**

Designed for web integration partner's store with eCom Checkout.

1. **Request to e-Com API** for obtaining a link to Checkout.

Designed for mobile integration partner's store with eCom Checkout.

### 2.2.1. eCom SDK fields description

| Field | Data type | Description | Validation |
|---|---|---|---|
| deepLinkUrl | string | Deep Link for redirecting payer to store's mobile app after payment | 0-9, a-Z, /:<br>starts with https:// |
| amount* | number | Order amount | 0-9, delimiter "." (point), value >=0 |
| currency | string | Order currency | Only "UAH" available |
| description | string | Order name | Max. 255 characters |
| partnerOrderId* | string | Unique order number in the Store | 0-9, a-Z, max 20 characters |
| merchantId* | Int32 | Unique eCom merchant identifier | |
| authType* | Int32 | Authorization type | Available options:<br>1 - Authorization<br>2 - PreAuthorization<br>4 - Verify |
| successRedirectUrl | string | Store's URL to which the customer will to be redirected after successful payment | 0-9, a-Z, /:<br>starts with https:// |
| failureRedirectUrl | string | Store's URL to which the customer will to be redirected after payment error | 0-9, a-Z, /:<br>starts with https:// |
| customParameters | string | Additional parameters | JSON with key-value parameters<br>F.e., filialId = "QR1234" |
| callbackUrl | string | Store's callback URL to which will be send payment result | 0-9, a-Z, /:<br>starts with https:// |
| cultureName | string | Response localization | Available options:<br>uk-UA (default) |
| phoneNumber | string | Phone number | Only Ukrainian numbers are allowed (for example, 0981234567, +38 098 1234567, (098) 123-45-67). |
| signature* | string | Signature | Base 64 format |
| keyHash* | string | Private key hash | 0-9, A-Z |

* - required fields.

**authType** values:

| authType | Description |
|---|---|
| Authorization | Ordinary authorization – the full amount of order is debited. |
| PreAuthorization | The amount of the order is debited, with the possibility of changing the amount when sending a request to pay method.<br>Refund (full or partial) of preauthorization is not allowed. To refund, you first need to send a post-authorization request (pay), and then send a refund request. |
| Verify | Deferred payment with card verification – 0 amount has been debited with tokenization of the card. For the generated token, the partner has the opportunity to debit any amount for the current order by calling the pay method. The token is disposable and is valid only within one order. |

### 2.2.2. eCom Javascript SDK

To use the **ECOM SDK**, you need to implement a script with a link to the project CDN:

```
<script src="{CDN HOST}/SDK/Source/ecom.sdk.js"></script>
```

, where {CDN HOST} depends on the environment:

- Test - https://sdk.ecom.test.vostok.bank/
- Production - https://sdk.ecom.vostok.bank/

***EcomSDK.eComPay*** is a class that provides methods for creating custom payment interfaces.

When using an instance of the ***EcomSDK.eComPay*** class, the client's redirection to checkout is requested by calling the corresponding checkout*()* method of this class.

An ***eComPay*** object is created using a constructor without parameters.

```
<script>
    const eComPay = new EcomSDK.CheckoutPay();
</script>
```

Next, you should initialize the object with order data. Script example:

```
<script>
    const eComPay = new EcomSDK.CheckoutPay();

    EcomPay.phoneNumber = "+380981234567";
    EcomPay.amount = 499.99;
    EcomPay.description = "Test payment";
    EcomPay.partnerOrderId = "ORD-773475757"
    EcomPay.merchantId = 1;
    EcomPay.authType = 1;
    EcomPay.successRedirectUrl = "https://core.test.api.bank.lan:6160/success.html";
    EcomPay.failureRedirectUrl = "https://core.test.api.bank.lan:6160/failure.html";
    EcomPay.callbackUrl = "http://merchant.com/callback";
    EcomPay.cultureName = "uk-UA";
    EcomPay.signature = "6156be9d17ba42ba9601d50668a0f11b";
    EcomPay.keyHash = "erhwhwer8h9weh8weh9r8h";
    EcomPay.customParameters = {
```

```
                param1: "some_data1",
                param2: "some_data2"
        }
    </script>
```

After the ***eComPay*** object is initialized with new values, you are ready to redirect the user to the checkout page.
To do this, call the checkout method of the ***eComPay*** object.

```
EcomPay.checkout();
```

After calling the *checkout()* method, the user is redirected to the checkout page and proceeds the payment following further instructions.

### 2.2.3. API e-Com SDK

If a partner wants to integrate e-Com Checkout in mobile applications, it is necessary to call the **create** method, filling all the fields described in the section eCom SDK fields description.

In response, a link (URL) to eCom Checkout will be received. Then a customer could be redirected by this link to proceed the payment.

## 2.3. eCom API methods

URL’s for sending requests:

- Test environment - https://api.ecom.test.vostok.bank
- Production environment - https://api.ecom.vostok.bank

**Please note!** All available input parameters in API methods should be sent in the request body and used to create the request's signature. If a parameter has option **Nullable = Yes** - it should be used with null value (f.e. *“cardToken”: null*).

### 2.3.1. /api/order/getOrder

Type: *POST*.

The method is used for obtaining order’s status and details.

Input parameters:

| Parameter | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| partnerOrderId | No | string | Unique Order ID in the merchant system | 0-9, a-Z |
| merchantId | No | Int32 | Unique e-Com merchant identifier | 0-9 |
| cultureName | No | string | Response localization | Currently only available uk-UA |

Output parameters:

| Parameter | Data type | Description |
|---|---|---|
| phoneNumber | Int64 | Customer's phone number |
| amount | number | Order amount |
| currency | string | Order currency |
| description | string | Order name |
| orderId | Int32 | Unique eCom order identifier |
| partnerOrderId | Int32 | Unique Store’s order identifier |
| orderStatusId | Int32 | Order status |
| authCode | string | Authorization code |
| authDateTime | datetime | Date and time of authorization |
| rrn | string | Unique authorization code |
| maskedCardNumber | string | Masked card number in the 6-4 format (111111******1111) |
| cardToken | string | Card token for authType = verify |
| errorCode | Int32 | Error code |
| errorMessage | string | Error text |

Error Codes:

| Code | Description |
|---|---|
| 200 | OK |
| 400 | Bad Request |
| 500 | Error |

Request example:

```
{
  "partnerOrderId": "417563-24-1",
  "merchantId": 1012,
  "cultureName": "uk-UA"
}
```

Response example:

```
{
  "phoneNumber": 380977777777,
  "amount": 1.25,
  "currency": "UAH",
  "description": "some description",
  "orderId": 33217,
  "partnerOrderId": "417563-24-1",
  "orderStatusId": 7,
  "authCode": null,
  "authDateTime": "2023-01-24T11:18:02",
  "rrn": null,
  "maskedCardNumber": "414951******7470",
  "cardToken": "6859b424-765c-4a74-91d6-9159133610ed",
  "errorCode": 0,
  "errorMessage": null
}
```

### 2.3.2. /api/order/pay

Type: *POST*.

Deferred payment method (after card verification).

Used to pay the different amount for order with *authType = PreAuthorization* or *Verify*.

If the partner's backend did not receive the final payment status (see Order statuses), it must periodically check the status of the order through the getOrder method until he receives one of the final payment statuses.

Input parameters:

| Parameter | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| partnerOrderId | No | string | Unique Order ID in the merchant system | 0-9, a-Z |
| merchantId | No | Int32 | Unique eCom merchant identifier | 0-9 |
| amount | No | number | Amount of payment | 0-9, delimiter "." (point), value >=0<br>For PostAuthorization, it cannot exceed the original order amount. |
| cardToken | Yes | string | Disposable card token | 0-9, a-Z<br>The token is valid only for the current order and is disposed of after successful payment. |
| cultureName | No | string | Response localization | Currently only available uk-UA |

Output parameters:

| Parameter | Data type | Description |
|---|---|---|
| authCode | string | Authorization code |

| Parameter | Data type | Description |
|---|---|---|
| authDateTime | datetime | Date and time of authorization |
| rrn | string | Unique authorization code |
| maskedCardNumber | string | Masked card number in 6-4 format (111111******1111) |
| errorCode | Int32 | Error code |
| errorMessage | string | Error text |

Error Codes:

| Code | Description |
|---|---|
| 200 | OK |
| 400 | Bad Request |
| 500 | Error |

Request example:

```
{
  "partnerOrderId": "417563-24-1",
  "merchantId": 1012,
  "amount": 1.25,
  "cardToken": "6859b424-765c-4a74-91d6-9159133610ed",
  "cultureName": "uk-UA"
}
```

Response example:

```
{
  "authCode": "837524",
  "authDateTime": "2023-01-24T11:19:18",
  "rrn": "302411697797",
  "maskedCardNumber": "414951******7470",
  "errorCode": 0,
  "errorMessage": null
}
```

#### 2.3.3. /api/order/cancel

Type - POST.

Cancellation/refund method for an order.

If the partner did not receive the final payment status (see.  Order statuses), he must periodically check the status of the order through the getOrder method until he receives one of the final payment statuses.

Restrictions on the calling of the method:

1. Within the current day, it is possible to make only one refund (full or partial), the amount of which should not exceed the purchase amount.
2. Starting from the next day, you can make full or up to 10 partial refunds, the total amount of which should not exceed the purchase amount.
3. Method is not available for purchases with the preAuthorization authorization type.

Input parameters:

| Parameter | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| partnerOrderId | No | string | Unique Order ID in the merchant system | 0-9, a-Z |
| merchantId | No | Int32 | Unique eCom merchant identifier | 0-9 |
| amount | No | number | Refund amount (for partial refunds) | 0-9, delimiter "." (point)<br>If not specified, a refund is made for the purchase amount |
| cultureName | No | string | Response localization | Currently only available uk-UA |
| settlementData | Yes | JSON | Used to refund on different card | |
| cardNumber | No | string | Card number (PAN) for refund | 0-9, 16 digits |
| expiryDate | No | string | Card's expiry date | Date format |

Output parameters:

| Parameter | Data type | Description |
|---|---|---|
| errorCode* | Int32 | Error code |
| errorMessage | string | Error text |

Error Codes:

| Code | Description |
|---|---|
| 200 | OK |
| 400 | Bad Request |
| 500 | Error |

Request example (refund on original card):

```
{
  "partnerOrderId": "417563-24-1",
  "merchantId": 1012,
  "amount": 1.10,
  "cultureName": "uk-UA"
}
```

Request example (refund on different card):

```
{
  "partnerOrderId": "417563-24-1",
  "merchantId": 1012,
  "amount": 1.25,
  "cultureName": "uk-UA",
  "settlementData": {
    "cardNumber": "1234567890123456",
    "expiryDate": "2024-10-20T00:00:00"
  }
}
```

Response example:

```
{
  "errorCode": 0,
  "errorMessage": null
}
```

### 2.3.4. /payInfo (callback)

Type: *POST*.

Callback-method of payment result, which sends to the Store's CallbackURL.

Output parameters:

| Parameter | Data type | Description |
|---|---|---|
| orderId | Int32 | Unique eCom order identifier |
| partnerOrderId | string | Unique partner order ID |
| orderStatusId | Int32 | Status (see. Order statuses) |
| authCode | string | Authorization code |
| authDateTime | datetime | Date and time of authorization |
| rrn | string | Unique authorization code |
| cardToken | string | One-time card token (only for authType = Verify) |
| maskedCardNumber | string | Masked card number in 6-4 format (111111******1111) |
| paymentMethod | string | Payment method (card, bvrpay, gpay, apay) |

Response example:

```
{
  "orderId": 33218,
  "partnerOrderId": "783722-24-1",
  "orderStatusId": 7,
  "authCode": null,
  "authDateTime": "2023-01-24T11:34:21",
  "rrn": null,
  "cardToken": "16ecf28a-ae46-4380-8c91-9af2fb790395",
  "maskedCardNumber": "414951******7470",
  "paymentMethod": "gpay"
}
```

### 2.3.5. /api/order/create

Type: POST.

The method is designed to register an order in the e-Com system and receive a link to go to Checkout in cases where JS integration of the library cannot be used (for example, for integration in mobile applications).

Input parameters (similar to the parameters when integrating through the JS eCom SDK):

| Field | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| deepLinkUrl | Yes | string | Deep Link for redirecting payer to store's mobile app after payment | 0-9, a-Z, /:<br>starts with https://<br>*null* if not used |
| amount | No | number | Order amount | 0-9, delimiter "." (point), value >=0 |
| currency | No | string | Order currency | Only "UAH" available |
| description | Yes | string | Order name | Max. 255 characters |
| partnerOrderId | No | string | Unique order number in the Store | 0-9, a-Z, max 20 characters |

| Field | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| merchantId | No | Int32 | Unique eCom merchant identifier | 0-9 |
| authType | No | Int32 | Authorization type | Avaliable options:<br>1 - Authorization<br>2 - PreAuthorization<br>4 - Verify |
| successRedirectUrl | Yes | string | Store's URL to which the customer will to be redirected after successful payment | 0-9, a-Z, /:<br>starts with https:// |
| failureRedirectUrl | Yes | string | Store's URL to which the customer will to be redirected after payment error | 0-9, a-Z, /:<br>starts with https:// |
| customParameters | Yes | string | Additional parameters | JSON with key-value parameters<br>F.e., filialId = "QR1234" |
| callbackUrl | Yes | string | Store's callback URL to which will be send payment result | 0-9, a-Z, /:<br>starts with https:// |
| cultureName | No | string | Response localization | Possible options:<br>uk-UA (default) |
| phoneNumber | Yes | string | Phone number | Only Ukrainian numbers are allowed (for example, 0981234567, +38 098 1234567, (098) 123-45-67). |
| signature | No | string | Signature | Base 64 format |
| keyHash | No | string | Private key hash | 0-9, A-Z |

**Please note, the order of fields in the request must be the same as in the description above.**

Output parameters:

| Parameter | Data type | Description |
|---|---|---|
| errorCode | Int32 | Error code |
| errorMessage | string | Error text |
| checkoutRedirectUrl | string | Checkout URL for redirect and proceed the payment |

Error Codes:

| Code | Description |
|---|---|
| 200 | OK |
| 400 | Bad Request |
| 500 | Error |

Request example:

```
{
  "deepLinkUrl": null,
  "amount": 3.75,
  "currency": "UAH",
  "description": "some description",
  "partnerOrderId": "ord12345",
  "merchantId": 1012,
  "authType": 1,
  "successRedirectUrl": "https://google.com",
  "failureRedirectUrl": "https://google.com",
  "customParameters": "{\"filialId\":\"QR1234\"}",
  "callbackUrl": "https://google.com",
  "cultureName": "uk-UA",
  "phoneNumber": 380977777777,
  "signature": "MTIzNDU=",
  "keyHash": "fhsk47sklkujf3u4whdgfjh43g"
}
```

Response example:

```
{
  "checkoutRedirectUrl":
"https://pay.test.bankvostok.com/Checkout/D23E60FAA58C13E801D86683595C7C49D1B62F1C5526CFFD4
E8D96710DF32B7B",
  "errorCode": 0,
  "errorMessage": null
}
```

#### 2.3.6. /api/order/approve

Type: POST.

The method is designed to register and check an order in e-Commerce if the partner uses direct integration with the e-Commerce API.

Input parameters:

| Field | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| cardDataType | No | string | Card Data Type for CardData value | Available options:<br>**1 - encryptedPAN** - encrypted card requisites (PAN, Expiry Date, CVV) - for PCIDSS certified partners only)*<br>**2 - tokenGuid** - e-Commerce card token GUID<br>**3 - GoogleToken** - Google Pay token<br>**4 - AppleToken** - Apple Pay token |
| cardData | No | string | Card requisites | Card requisite according to cardDataType: |

| Field | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| | | | | For GPay/APay the JSON-token, received from Google/Apple API, must be filled |
| amount | No | number | Order amount | 0-9, delimiter "." (point), value >=0 |
| browserData (JSON object) | Yes | JSON | Browser data for 3DS check | |
| browserData.browserAcceptHeader | No | string | Browser header | Max 2048 characters |
| browserData.browserColorDepth | No | string | Browser color depth (bit) | Available values: 1\|4\|8\|15\|16\|24\|32\|48 |
| browserData.browserJavaEnabled | No | string | Java is enabled | Available values: Y\|N |
| browserData.browserJavascriptEnabled | No | string | Javascript is enabled | Available values: Y\|N |
| browserData.browserLanguage | No | string | Browser language | Max 48 characters |
| browserData.browserScreenHeight | No | string | Browser screen resolution height | 0-9, Min 1, Max 999999 |
| browserData.browserScreenWidth | No | string | Browser screen resolution width | 0-9, Min 1, Max 999999 |
| browserData.browserTZ | No | string | Browser timezone | Min 1, Max 5 characters |
| browserData.browserUserAgent | No | string | Browser User Agent | Max 2048 characters |
| eventCallbackUrl | Yes | string | Store's URL for redirecting user after 3DS verification | 0-9, a-Z, /: starts with https:// |
| currency | No | string | Order currency | Only "UAH" available |
| description | Yes | string | Order description | Max. 255 characters |
| partnerOrderId | No | Int32 | Unique order number in the Store | 0-9, a-Z, max 20 characters |
| merchantid | No | Int32 | Unique eCom merchant identifier | 0-9 |
| cultureName | No | string | Response localization | Available options: uk-UA (default) |
| customParameters | Yes | JSON | Additional parameters | JSON with key-value parameters F.e., filialId = "QR1234" |

*For using **cardDataType = encryptedPAN** you need to:

1. Create JSON with card requisites:
```
{
  "cardNumber": "1234567890123456",
  "expiryDate": "2024-10-20T00:00:00",
  "cvv": "123"
}
```
2. Encrypt obtained JSON, using the specific public key, provided by Bank..
3. Send encrypted data at BASE64 format in **cardData** parameter value.

### C# encryption example

```
string publicKey = "-----BEGIN PUBLIC KEY-----TEST-----END PUBLIC KEY-----";
RSA rsa = RSA.Create();
rsa.ImportFromPem(publicKey);
string cardData =
"{\"cardNumber\":\"4999999999990011\",\"expiryDate\":\"2024-10-20T00:00:00\",\"cvv\":\"000\
"}";
byte[] cardDataBytes = Encoding.UTF8.GetBytes(cardData);
byte[] encryptedCardData = rsa.Encrypt(cardDataBytes, RSAEncryptionPadding.Pkcs1);
return Convert.ToBase64String(encryptedCardData);
```

Output parameters:

| Parameter | Data type | Description |
|---|---|---|
| errorCode | Int32 | Error code |
| errorMessage | string | Error text |
| orderId | Int32 | Unique eCom order identifier |
| emv3DSVersion | Int32 | 3DS verification check flag:<br>0 - 3DS not available or 3DS v.2 Frictionless Flow (without additional verification)<br>2 - 3DS v.2 |
| challengeUrl | string | URL банку емітента для проведення 3DSecure перевірки (для emv3DSVersion = 2) |

Error Codes:

| Code | Description |
|---|---|
| 200 | OK |
| 400 | Bad Request |
| 500 | Error |

Request example for **encryptedPAN**:

```
{
    "cardDataType": 1,
    "cardData":
"B2uxk6IpxqcJzD89I6tR+SXuX46YXafUX8B4m41ijVCfXz++d6LTpnQulTMmKsZJ2ksIvwfCgzrwsHCyFHJ+ZXdglL
HB9W0cyKgYWlM+euqu3cHoBbj3NMBt0XuZvFk8XDDHvzm48ACEzGrBJ+RJCwoiuCwHhO2IdIO0msYXJA+wJslg5X9Ib
V1jRN4HTMwfd1648TYzT4npWE56g4ziynLNMsCTVYFNDy45WpUW0pYOcwohhZofM+6KJ8aH8I7V6W69o5TBxxu3ZMxU
Sgy3Yp2LV1z5pFikWHhWmO03No7++VLsZI5FH6nFqeX7BGnbIT+tXm62K0COHQ39wsvUdw==",
    "amount": 5,
    "browserData": {
        "browserAcceptHeader":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,appl
ication/signed-exchange;v=b3;q=0.9",
        "browserColorDepth": 24,
        "browserJavaEnabled": "Y",
        "browserJavaScriptEnabled": "Y",
        "browserLanguage": "en-US",
        "browserScreenHeight": 1080,
        "browserScreenWidth": 1920,
        "browserTZ": -120,
        "browserUserAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"
    },
    "eventCallbackUrl": "https://google.com",
    "currency": "UAH",
    "description": "testDescription",
    "partnerOrderId": "testOrder",
    "merchantId": 2003,
    "cultureName": "uk-UA",
    "customParameters": {
        "additionalProp1": "string1",
        "additionalProp2": "string2",
        "additionalProp3": "string3"
```

Request example for **tokenGuid**:

```
{
    "cardDataType": 2,
    "cardData": "87c9ddfc-9cf4-4d7b-b120-a32e7df66441",
    "amount": 5,
    "browserData": {
        "browserAcceptHeader":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,appl
ication/signed-exchange;v=b3;q=0.9",
        "browserColorDepth": 24,
        "browserJavaEnabled": "Y",
        "browserJavaScriptEnabled": "Y",
        "browserLanguage": "en-US",
        "browserScreenHeight": 1080,
        "browserScreenWidth": 1920,
        "browserTZ": -120,
        "browserUserAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"
    },
    "eventCallbackUrl": "https://google.com",
    "currency": "UAH",
    "description": "testDescription",
    "partnerOrderId": "testOrder",
    "merchantId": 2003,
    "cultureName": "uk-UA",
    "customParameters": {
        "additionalProp1": "string1",
        "additionalProp2": "string2",
        "additionalProp3": "string3"
    }
}   }
}
```

Request example for **GoogleToken**:

```
{
    "cardDataType": 3,
    "cardData":
"{\"signature\":\"MEUCIQC1d/iAa9tB5IBzsbHQfSfWFgVkhp9RHYEDtzITSOgrHwIgLox0XPEQN8BlNBMsaQiO+
IPPRMdQBzBvRInj3UqdFKg\\u003d\",\"intermediateSigningKey\":{\"signedKey\":\"{\\\"keyValue\\
\":\\\"MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEJvO354LewFwxAIIvQd4W3n7ifYh/Xa5MwAOgxH8lWaFbwNBD
gjO/K3fffwvUpkgxDXQ+s8kMz7JtpaPrsZgdjQ\\\u003d\\\u003d\\\",\\\"keyExpiration\\\":\\\"1684
993843083\\\"}\",\"signatures\":[\"MEUCIDGWgiDkiJ7K8km4Tiq0cmU0iLgnDTf1r68RhVkqdl3OAiEAklXh
1V87PeaYi3CVKfDrGGOwNMgKV0aDKBFDMQl4A7E\\u003d\"]},\"protocolVersion\":\"ECv2\",\"signedMes
sage\":\"{\\\"encryptedMessage\\\":\\\"i6WvKWrhv+DYLPIVthmXMS1tAcITL593A8+aBqoxlkORFvitN/g8
Us3JXw5AAzGJPbQu/Y+K2eKUjrL5jWFpZle+woJynJpBmpPBCWSoJysiep2DYGbcqn1HIKZKpL9IhUO0Z/MHClmpzLP
oQ9EXHiTigzwvZb/3eez342wYK45L96hOd9sADxlKp/Uf/bHJ+SZaolLw8veTRwXBmUmVvGWdDtYMcnLGlNEKJa07Hz
mfp9C31ZV+uQ41qQbCA5KKmllA8GzLIUsfgvsm1Eetxrl8YzVHgW4hle4wk0vOy237NKeR3+gYmuViB7QZdwmhj3AKh
JfKeNhRfO0n8hssgsM5ugKvlQMx/BnGnuhrfddHTg1LtU/J/3x/J8aqee/0ZRSolcAzRF00ekS45Q0a6NhjqAugjE+J
6O4wSidNEh6YgR92gbYzDpjyCpGsfXKxrTcKAfrreTW3R/YXhrtETvtpsOa+fBt9OSnkMZr/FDSAnqh1VjsbaXGhZrf
7TvpNnSUsyFqMdfj9H9Zehu0ZeQ8S4FKe4cZ4Co8gri4R7hLevYssfZEtvt173c15gf3T9+A9\\\",\\\"ephemeral
PublicKey\\\":\\\"BHGKvFrwVmhnrYbFB60Lve9FpL5DU/0+T3OQGiROTY/IZMWLOIub02T6/cJuIpMMLp3dDjc7h
jsnlA711WjUzF0\\\\u003d\\\",\\\"tag\\\":\\\"KRQgEZxjL0CSKpGI0fyUE7BimR2cKUgm68EuXCO1fNE\\\\
u003d\\\"}\"}",
    "amount": 5,
    "browserData": {
        "browserAcceptHeader":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,appl
ication/signed-exchange;v=b3;q=0.9",
        "browserColorDepth": 24,
        "browserJavaEnabled": "Y",
        "browserJavaScriptEnabled": "Y",
        "browserLanguage": "en-US",
        "browserScreenHeight": 1080,
        "browserScreenWidth": 1920,
        "browserTZ": -120,
        "browserUserAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"
    },
    "eventCallbackUrl": "https://google.com",
    "currency": "UAH",
    "description": "testDescription",
    "partnerOrderId": "testOrder",
    "merchantId": 2003,
    "cultureName": "uk-UA",
    "customParameters": {
        "additionalProp1": "string1",
        "additionalProp2": "string2",
        "additionalProp3": "string3"
    }
}
```

Request example for **AppleToken**:

```
{
    "cardDataType": 4,
    "cardData":
"{\"paymentData\":{\"data\":\"yPtlfNvP5VE/srwkoc1xvpVf54JDoTBnzk8duUjeQCC7rekMetgx2ZW/3pCFK
1WoBHJrLCZ8o8f5vIrv7yuoQoUP9B0qKByAi/+DpCA6VyEs8u5hhK/BIir/S4ov04y+/2V7S/E2BEEDDciyZAOP9oyH
G8N+otJFUDIw6SnOR3cttsRUbJQjCVUiScMuqcUO09yYJZFs/p36uh0D8uKTi9iHB1+8tRqzPVk/r/pEapt98cMFEx7
DGhBJckCpr26AQXvIMhGVunDlgnI/R1Z6qg1adQR3Be9m7ovQgLpj54UbvEvCW9/YCtSZrq1l0xRU53JC8i2doRaLx7
TT9E06pBjRfvwHOKaoLEntUzZWxxaV1SjjV5n3eG3U5C3yMEGuPnpTjhEpn09sB4I=\",\"signature\":\"MIAGCS
qGSIb3DQEHAqCAMIACAQExDTALBglghkgBZQMEAgEwgAYJKoZIhvcNAQcBAACggDCCA+MwggOIoAMCAQICCEwwQUlRn
VQ2MAoGCCqGSM49BAMCMHoxLjAsBgNVBAMMJUFwcGxlIEFwcGxpY2F0aW9uIEludGVncmF0aW9uIENBIC0gRzMxJjAk
BgNVBAsMHUFwcGxlIENlcnRpZmljYXRpb24gQXV0aG9yaXR5MRMwEQYDVQQKDApBcHBsZSBJbmMuMQswCQYDVQQGEwJ
VUzAeFw0xOTA1MTgwMTMyNTdaFw0yNDA1MTYwMTMyNTdaMF8xJTAjBgNVBAMMHGVjYy1zbXAtYnJva2VyLXNpZ25fVU
M0LVBST0QxFDASBgNVBAsMC2lPUyBTeXN0ZW1zMRMwEQYDVQQKDApBcHBsZSBJbmMuMQswCQYDVQQGEwJVUzBZMBMGB
yqGSM49AgEGCCqGSM49AwEHA0IABMIVd+3r1seyIY9o3XCQoSGNx7C9bywoPYRgldlK9KVBG4NCDtgR80B+gzMfHFTD
9+syINa61dTv9JKJiT58DxOjggIRMIICDTAMBgNVHRMBAf8EAjAAMB8GA1UdIwQYMBaAFCPyScRPk+TvJ+bE9ihsP6K
7/S5LMEUGCCsGAQUFBwEBBDkwNzA1BggrBgEFBQcwAYYpaHR0cDovL29jc3AuYXBwbGUuY29te29jc3AwNC1hcHBsZW
FpY2EzMDIwggEdBfNVHSAEggEUMIIBEDCCAgwGCSqGSIb3Y2QFATCB/jCBwwYIKwYBBQUHAgIwgbYMgbNSZWxpYW5jZ
SBvbiB0aGlzIGNlcnRpZmljYXRlIGJ5IGFueSBwYXJ0eSBhc3N1bWVzIGFjY2VwdGFuY2Ugb2YgdGhlIHRoZW4gYXBw
bGljYWJsZSBzdGFuZGFyZCB0ZXJtcyBhbmQgY29uZGl0aW9ucyBvZiB1c2UsIGNlcnRpZmljYXRlIHBvbGljeSBhbmQ
gY2VydGlmaWNhdGlvbiBwcmFjdGljZSBzdGF0ZW1lbnRzLjA2BggrBgEFBQcCARYqaHR0cDovL3d3dy5hcHBsZS5jb2
0vY2VydGlmaWNhdGVhdXRob3JpdHkvMDQGA1UdHwQtMCswKaAnoCWGI2h0dHA6Ly9jcmwuYXBwbGUuY29tL2FwcGxlY
WljYTMuY3JsMB0GA1UdDgQWBBSUV9tv1XSBhomJdi9+V4UH55tYJDAOBgNVHQ8BAf8EBAMCB4AwDwYJKoZIhvdjZAYd
BAIFADAKBggqhkjOPQQDAgNJADBGAiEAvglXH+ceHnNbVeWvrLTHL+tEXzAYUiLHJRACth69b1UCIQDRizUKXdbdbrF
0YDWxHrLOh8+j5q9svYOAiQ3ILN2qYzCCAu4wggJ1oAMCAQICCEltL786mNqXMAoGCCqGSM49BAMCMGcxGzAZBgNVBA
MMEkFwcGxlIFJvb3QgQ0EgLSBHMzEmMCQGA1UECwwdQXBwbGUgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkxEzARBgNVB
AoMCkFwcGxlIEluYy4xCzAJBgNVBAYTAlVTMB4XDTE0MDUwNjIzNDYzMFoXDTI5MDUwNjIzNDYzMFowejEuMCwGA1UE
AwwlQXBwbGUgQXBwbGljYXRpb24gSW50ZWdyYXRpb24gQ0EgLSBHMzEmMCQGA1UECwwdQXBwbGUgQ2VydGlmaWNhdGl
vbiBBdXRob3JpdHkxEzARBgNVBAoMCkFwcGxlIEluYy4xCzAJBgNVBAYTAlVTMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQ
cDQgAE8BcRhBnXZIXVGl4lgQd26ICi7957rk3gjfxLk+EzVtVmWzWuItCXdg0iTnu6CP12F86Iy3a7ZnC+yOgphP9UR
aOB9zCB9DBGBggrBgEFBQcBAQQ6MDgwNgYIKwYBBQUHMAGGKmh0dHA6Ly9vY3NwLmFwcGxlLmNvbS9vY3NwMDQtYXBw
bGVyb290Y2FnMzAdBgNVHQ4EFgQUI/JJxE+T5O8n5sT2KGw/orv9LkswDwYDVR0TAQH/BAUwAwEB/zAfBgNVHSMEGDA
WgBS7sN4dWDOImqSKmd6+veuv2sskqzA3BgNVHR8EMDAuMCygKqAohiZodHRwOi8vY3JsLmFwcGxlLmNvbS9hcHBsZX
Jvb3RjYWczLmNybDAOBgNVHQ8BAf8EBAMCAQYwEAYKKoZIhvdjZAYCDgQCBQAwCgYIKoZIzj0EAwIDZwAwZAIwOs9yg
1EWmbGG+zXDVspiv/QX7dkPdU2ijr7xnIFeQreJ+Jj3m1mfmNVBDY+d6cL+AjAyLdVEIbCjBXdsXfM4O5Bn/Rd8LCFt
lk/GcmmCEm9U+Hp9G5nLmwmJIWEGmQ8Jkh0AADGCAYgwggGEAgEBMIGGMHoxLjAsBgNVBAMMJUFwcGxlIEFwcGxpY2F
0aW9uIEludGVncmF0aW9uIENBIC0gRzMxJjAkBgNVBAsMHUFwcGxlIENlcnRpZmljYXRpb24gQXV0aG9yaXR5MRMwEQ
YDVQQKDApBcHBsZSBJbmMuMQswCQYDVQQGEwJVUwIITDBBSVGdVDYwCwYJYIZIAWUDBAIBoIGTMBgGCSqGSIb3DQEJA
zELBgkqhkiG9w0BBwEwHAYJKoZIhvcNAQkFMQ8XDTIzMDUxNzExMzUyMVowKAYJKoZIhvcNAQk0MRswGTALBglghkgB
ZQMEAgGhCgYIKoZIzj0EAwIwLwYJKoZIhvcNAQkEMSIEIPxVyIZ10pF3eNxU+V2o1Fz9LKWGg9HeVdgCQaKtgxJ2MAo
GCCqGSM49BAMCBEcwRQIhAP3rLi4IJWK10bOazilOrAR0N96KJqxvP0qhY/UFDh/TAiAB2iqd2yho+o7wHswP1xkC/a
aypZZnjapbqUzLx2Gm8wAAAAAAAA==\",\"header\":{\"publicKeyHash\":\"xpXrfDOi0rTreU83BmzHWUIdLk
C3QO0C6hJBOeiFmsc=\",\"ephemeralPublicKey\":\"MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEWu2wrb0rj
EKjYHhNr8aBXYgwoeT61+KT/RZ3cpVMh1cSBB1ByvdfERUqKbmOojh7Cgbv9LkKGiqWP0pquM/48Q==\",\"transac
tionId\":\"38aa8bfbd9811f6874ba17c9c6d0207d731b7a1a529ecfe64176bf07e3279fc8\"},\"version\":
\"EC_v1\"},\"paymentMethod\":{\"displayName\":\"MasterCard0241\",\"network\":\"MasterCard\"
,\"type\":\"debit\"},\"transactionIdentifier\":\"38aa8bfbd9811f6874ba17c9c6d0207d731b7a1a52
9ecfe64176bf07e3279fc8\"}",
    "amount": 5,
    "browserData": {
        "browserAcceptHeader":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,appl
ication/signed-exchange;v=b3;q=0.9",
        "browserColorDepth": 24,
        "browserJavaEnabled": "Y",
        "browserJavaScriptEnabled": "Y",
        "browserLanguage": "en-US",
        "browserScreenHeight": 1080,
        "browserScreenWidth": 1920,
        "browserTZ": -120,
        "browserUserAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"
    },
    "eventCallbackUrl": "https://google.com",
    "currency": "UAH",
    "description": "testDescription",
    "partnerOrderId": "testOrder",
    "merchantId": 2003,
    "cultureName": "uk-UA",
    "customParameters": {
        "additionalProp1": "string1",
        "additionalProp2": "string2",
        "additionalProp3": "string3"
    }
}
```

Response example:

```
{
  "emv3DSVersion": 0,
  "challengeUrl": null,
  "errorCode": 0,
  "errorMessage": null
}
```

### 2.3.7. /api/order/confirm

Type: POST.

The method is used to confirm payment in e-Commerce if the partner uses direct integration with the e-Commerce API and receives a successful response from approve method.

If the partner's backend did not receive the final payment status (see Order statuses), it must periodically check the status of the order through the getOrder method until he receives one of the final payment statuses.

Input parameters:

| **Field** | **Nullable** | **Data type** | **Description** | **Validation** |
|---|---|---|---|---|
| partnerOrderId | No | string | Unique Order ID in the merchant system | 0-9, a-Z |
| merchantId | No | Int32 | Unique eCom merchant identifier | 0-9 |

* - required fields.

Output parameters:

| **Parameter** | **Data type** | **Description** |
|---|---|---|
| orderId | int32 | Unique eCom order identifier |
| orderStatusId | Int32 | Order status |
| authCode | string | Authorization code |
| authDateTime | datetime | Date and time of authorization |
| rrn | string | Unique authorization code |
| maskedCardNumber | string | Masked card number in 6-4 format (111111******1111) |
| errorCode | Int32 | Error code |
| errorMessage | string | Error text |

Error Codes:

| **Code** | **Description** |
|---|---|
| 200 | OK |
| 400 | Bad Request |
| 500 | Error |

Request example:

```
{
  "partnerOrderId": "te2GwsmvQ9",
  "merchantId": 1012
}
```

Response example:

```
{
  "orderStatusId": 2,
  "authCode": "470936",
  "authDateTime": "2023-03-20T13:29:15",
  "rrn": "307913850606",
  "maskedCardNumber": "414951******7470",
  "errorCode": 0,
  "errorMessage": null
}
```

#### 2.3.8. /api/cardToken/getUrl

The method is designed to obtain a link to the card tokenization.

It allows users to tokenize the card both for a single payment of a specific order (in case of filling *partnerOrderId*) or to create a permanent token for paying future orders (with an empty *partnerOrderId*).

For this method, a different format of the signature validation string for the requests is used. It should be formed like this:

**{merchantId}|{partnerCardOwnerId}|{callbackUrl}|{partnerOrderId}**

Input parameters:

| Parameter | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| partnerOrderId | Yes | string | Unique order code in the store | 0-9, a-Z, max 20 characters |
| merchantId | No | Int32 | Merchant Id in the e-Com system | 0-9 |
| parnetCardOwnerId | No | string | Unique customer code in the partner system, to which the token will be linked | 0-9, a-Z, max  50 characters |
| cardAlias | No | string | Card name | Max 50 characters |
| successRedirectUrl | Yes | string | URL of the store, to which the customer should be redirected after successful tokenization | 0-9, a-Z, /:<br>starts with https://<br>If the eCom merchant's requisites specify a default **successRedirectUrl**, then it is used instead of the one passed here<br>null - if not used |
| failureRedirectUrl | Yes | string | URL of the store, to which the customer should be redirected after an error during tokenization | 0-9, a-Z, /:<br>starts with https://<br>If the eCom merchant's requisites specify a default **failureRedirectUrl**, |

| Parameter | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| | | | | then it is used instead of the one passed here<br>null - if not used |
| callbackUrl | Yes | string | URL of the store's API, to which the result and data of tokenization should be passed | 0-9, a-Z, /:<br>starts with https://<br>If the eCom merchant's requisites specify a default **callbackUrl**, then it is used instead of the one passed here<br>null - if not used |
| cultureName | No | string | Localization of the response | Possible options: uk-UA |
| signature | No | string | e-signature of the request | 0-9, A-Z |
| keyHash | No | string | Hash of the client's private key | BASE64 format |

Output parameters:

| Parameter | Data type | Description |
|---|---|---|
| tokenizationUrl | string | Link to the tokenization page |
| errorCode | Int32 | Error code |
| errorMessage | string | Error text |

Example of a request:

```
{
    "partnerOrderId": null,
    "merchantId": 7,
    "partnerCardOwnerId": "testCardOwner",
    "cardAlias": "testCard",
    "successRedirectUrl": "https://google.com",
    "failureRedirectUrl": "https://google.com",
    "callbackUrl": "https://google.com",
    "cultureName": "uk-UA",
    "signature": "MTIzNDU=",
    "keyHash": "fj44kfe7ed5f4fjj419a3756hhf686f2d1594840b087"
  }
```

Example of a response:

```
{

    "tokenizationUrl":
"https://wallet.api.test.bankvostok.com?id=15d21bfd11dd4da1ac69e365133b6f9f",

    "errorCode": 0,

    "errorMessage": null

}
```

### 2.3.9. /api/cardToken/getAll

The method is used to obtain a list of all tokenized cards by the unique customer identifier in the partner system (*partnerCardOwnerId*).

Input parameters:

| Parameter | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| parnetCardOwnerId | No | string | Unique customer code in the partner system, to which the token will be linked | 0-9, a-Z, max 50 characters |
| merchantId | No | Int32 | Merchant Id in the e-Com system | 0-9 |
| cultureName | No | string | Localization of the response | Possible options: uk-UA |

Output parameters:

| Pararmeter | Data type | Description |
|---|---|---|
| cardTokens | Object | An array with a list of cards |
| cardTokenGuid | string | Token (Guid) of the card in e-Com |
| cardAlias | string | Card name |
| maskedCardNumber | string | Masked card number (6-4) |
| paymentSystem | string | Card payment system (MasterCard, Visa) |
| partnerOrderId | string | Order code, for which the card is tokenized (for cases when tokenization is done for a specific order) |
| errorCode | Int32 | Error code |
| errorMessage | string | Error text |

Example of a request:

```
{
    "partnerCardOwnerId": "Client_001231",
    "merchantId": 7,
    "cultureName": "uk-UA"
}
```

Example of a response:

```
{
    "cardTokens": [
      {
        "cardTokenGuid": "c83782ab-b6e5-4771-a575-f1375599ddfe",
        "cardAlias": "Моя картка",
        "maskedCardNumber": "499999******0011",
        "paymentSystem": "Visa",
        "partnerOrderId": "27384623"
      },
      {
        "cardTokenGuid": "608e75e6-3bb8-4466-8336-b8f4d7c71e78",
        "cardAlias": "Замовлення 52634234",
        "maskedCardNumber": "499999******0011",
        "paymentSystem": "Visa",
        "partnerOrderId": "52634234"
      },
      {
        "cardTokenGuid": "8366ae70-569e-48e8-b0e4-194d65969e86",
        "cardAlias": "Основна",
        "maskedCardNumber": "499999******0011",
        "paymentSystem": "Visa",
        "partnerOrderId": null
      },
      {
        "cardTokenGuid": "0bc03187-da27-4e87-963e-1b415fefac87",
        "cardAlias": "ЗП",
        "maskedCardNumber": "499999******0011",
        "paymentSystem": "Visa",
        "partnerOrderId": null
      }
    ],
    "errorCode": 0,
    "errorMessage": null
}
```

### 2.3.10. /api/cardToken/updateAlias

The method is used to change the name of a previously tokenized card.

Input parameters:

| Parameter | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| cardTokenGuid | No | string | Token (Guid) of the card in e-Com | Guid-format |
| cardAlias | No | string | Card name | Max 50 characters |
| merchantId | No | Int32 | Merchant Id in the e-Com system | 0-9 |
| cultureName | No | string | Localization of the response | Possible options: uk-UA |

Output parameters:

| Parameter | Data type | Description |
|---|---|---|
| errorCode | Int32 | Error code |
| errorMessage | string | Error text |

Example of a request:

```
{
    "cardTokenGuid": "8AFFB98A-160D-4C78-A337-D5D53800355A",
    "cardAlias": "My new name",
    "merchantId": 7,
    "cultureName": "uk-UA"
}
```

Example of a response:

```
{
    "errorCode": 0,
    "errorMessage": null
}
```

### 2.3.11. /api/cardToken/delete

The method is used to delete a tokenized card.

Input parameters:

| Parameter | Nullable | Data type | Description | Validation |
|---|---|---|---|---|
| cardTokenGuid | No | string | Token (Guid) of the card in e-Com | Guid-format |
| merchantId | No | Int32 | Merchant Id in the e-Com system | 0-9 |
| cultureName | No | string | Localization of the response | Possible options: uk-UA |

Output parameters:

| Parameter | Data type | Description |
|---|---|---|
| errorCode | Int32 | Error code |
| errorMessage | string | Error text |

Example of a request:

```
{
    "cardTokenGuid": "8AFFB98A-160D-4C78-A337-D5D53800355A",
    "merchantId": 7,
    "cultureName": "uk-UA"
}
```

Example of a response:

```
{
    "errorCode": 0,
    "errorMessage": null
}
```

## 2.4. Google Pay

### 2.4.1. e-Commerce Hosted Checkout integration

Google Pay™ – is a quick and easy way that allows you to make payments by card details without entering them for each payment. When using our Hosted Checkout partner doesn't need any additional Google Pay integrations - Google Pay button is available in Checkout by default.

If you want to disable GPay payment method - please contact our support e-Commerce@bankvostok.com.ua

### 2.4.2. Direct integration with Google Pay API

If partner chooses direct integration with Google Pay API (f.e., to display GPay button in store's card), he must:

1. Register with Google Pay API in Google Pay & Wallet Console
2. Store's site must use HTTPS and support the TLS protocol.
3. Read and proceed all instructions in Google documentation:
   - for mobile applications
   - for web
   - brand guidelines for mobile applications
   - brand guidelines for web
4. Configure Google Pay initialization script (sample):
   - *allowPaymentMethods : CARD*
   - *tokenizationSpecification = { "type": "PAYMENT_GATEWAY"}*
   - *allowedCardNetworks = ['MASTERCARD', 'VISA'];*
   - *allowedCardAuthMethods = ['PAN_ONLY', 'CRYPTOGRAM_3DS'];*
   - *gateway = bankvostok*
   - *gatewayMerchantId* - unique merchant identifier, received during registration in e-Commerce (merchantId).
5. Verify store's site or mobile application in Google Pay & Wallet Console

Payment flow during direct integration with Google Pay API:

1. The customer chooses a card in Google Pay wallet.
2. The store receives encrypted JSON-token with the chosen card.
3. The store's backend sends a request to check the payment and 3DS verification availability (approve). Google Pay token should be filled in *cardData* field (in JSON format).
4. e-Commerce decrypts received token and returns response with availability of 3DSecure verification (*emv3DSVersion*).
5. If *emv3DSVersion* is not equal *0* - store should redirect customer to received URL to proceed the 3DSecure verification.
6. Store sends a confirmation request (confirm).
7. e-Commerce authorizes the payment.
8. The store displays the payment result to the customer.

### 2.5. Apple Pay

#### 2.5.1. e-Commerce Hosted Checkout integration

Apple Pay™ – is a quick and easy way that allows you to make payments by card details without entering them for each payment. When using our Hosted Checkout partner doesn't need any additional Apple Pay integrations - Apple Pay button is available in Checkout by default.

If you want to disable APay payment method - please contact our support e-Commerce@bankvostok.com.ua

#### 2.5.2. Direct integration with Apple Pay API

If partner chooses direct integration with Google Pay API (f.e., to display GPay button in store's card), he must:

1. Register an Apple Developer Account.
2. Store's site must use HTTPS and support the TLS protocol.
3. Using Apple Developer Account, create Apple MerchantId.
   - ☐ Log in to Apple Developer Account..
   - ☐ Open *«Certificates, Identifiers & Profiles».*
   - ☐ In the «Identifiers» chapter choose *«Merchant IDs».*
   - ☐ Create a new Merchant ID by clicking "+" at the top right corner.
   - ☐ Fill the fields:

     **Description** - description of your store

     **Idenifier** - merchant identifier like **merchant.ua.shop**, where ua.shop - mirrored name of the store's site.
   - ☐ Click *«Continue».*
   - ☐ Click *«Register»* to confirm and save merchant.
4. Send an email to e-Commerce@bankvostok.com.ua with followed data:
   - Apple MerchantId
   - Contact person mobile number for receiving shared password from bank (will be used to certificates encryption/decryption below).
5. The bank sends a shared password to the received mobile number.
6. In response to the letter which was sent above, the bank will send CSR (certificate signing request) in the archive, protected with the shared password.
7. Using a CSR partner must generate Apple Pay Payment Processing Certificate via Apple Developer Account.
   - ☐ Log in to Apple Developer Account..
   - ☐ Open *«Certificates, Identifiers & Profiles».*
   - ☐ In the «Identifiers» chapter choose *«Merchant IDs».*
   - ☐ Select the created Merchant ID and click *«Edit».*
   - ☐ In *Apple Pay Payment Processing Certificate*, click *«Create Certificate».*
   - ☐ Click *«Continue»* at the next step.
   - ☐ Upload the CSR-file by clicking *«Choose File»*, then click *«Continue».*
   - ☐ Download generated certificate.
8. Send generated certificate (*.cer) to e-Commerce@bankvostok.com.ua in a shared password protected archive.
9. Verify store's domain name via Apple Developer Account.

- ☐ Log in to Apple Developer Account..
- ☐ Open *«Certificates, Identifiers & Profiles».*
- ☐ In the «Identifiers» chapter choose *«Merchant IDs».*
- ☐ Select the created Merchant ID and click *«Add domain».*
- ☐ Fill the store's domain URL and click *«Save».*
- ☐ Download txt-file for verification by clicking *«Download».*
- ☐ Host received txt-file by URL https://yourdomain.com/.well-known/apple-developer-merchantid-domain-association.txt, where *yourdomain.com* - your store's domain URL.
- ☐ Click *«Verify».*

In case of successful verification your domain will change state to *«Verified»* in the Apple Developer Account and you may check domain expiry date next to it.

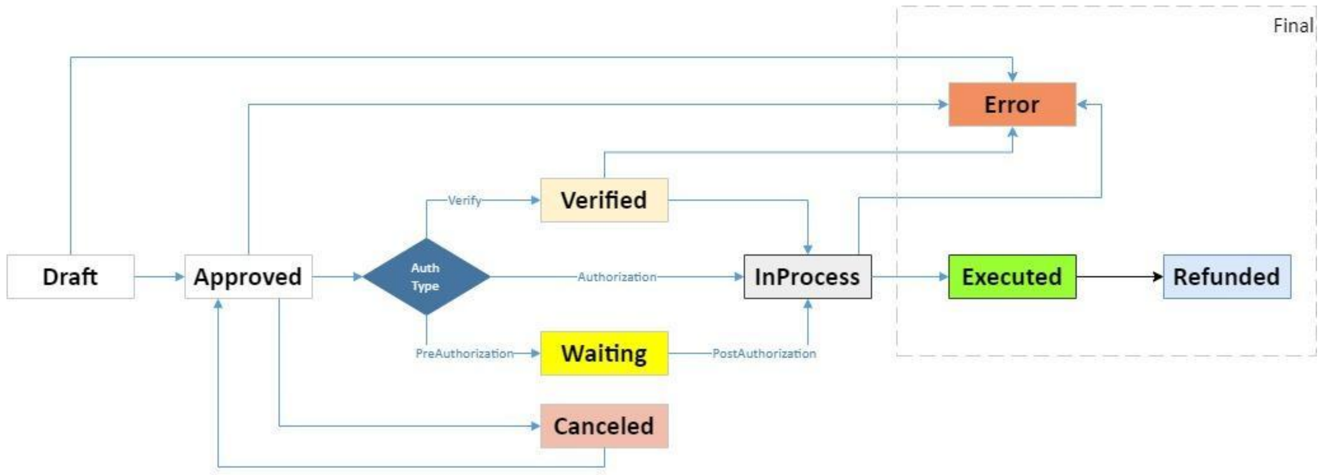
Процес проведення оплати при прямій інтеграції через Apple Pay API виглядає наступним чином:

1. The customer chooses a card in the Apple Pay wallet.
2. The store validates Apple Pay Session.
3. The store receives encrypted JSON-token with the chosen card.
4. The store's backend sends a request to check the payment and 3DS verification availability (approve). Apple Pay token should be filled in *cardData* field (in JSON format).
5. e-Commerce decrypts received token and returns response with availability of 3DSecure verification (*emv3DSVersion*).
6. If *emv3DSVersion* is not equal *0* - store should redirect customer to received URL to proceed the 3DSecure verification.
7. Store sends a confirmation request (confirm).
8. e-Commerce authorizes the payment.
9. The store displays the payment result to the customer.

## 2.6. Order statuses

| Code | Name | Description | Final status |
|---|---|---|---|
| 1 | Draft | New order<br>*Set when saving an order in e-Commerce* | No |
| 2 | Executed | Successfully paid<br>*Set after successful authorization of payment for the order (on the confirm method) for authType = "Authorization" or "PostAuthorization"* | Yes |
| 3 | Approved | Push confirmation sent<br>*Installed after successful execution of push sending for confirmation* | No |
| 4 | Canceled | Canceled order<br>*If payment is rejected from the order confirmation screen in the BVR application* | No |
| 5 | Refunded | The order is refunded<br>*After successful refund of the order* | Yes |
| 6 | Waiting | Pre-authorized order<br>*Set after successful pre-authorization of payment for the order for authType = "PreAuthorization"* | No |
| 7 | Verified | Verified order<br>*Set after successful pre-authorization of payment for the order for authType = "Verify"* | No |
| 8 | Error | Payment error<br>*If you encounter errors when paying for the order* | Yes |
| 9 | InProcess | The order is in the process of payment confirmation<br>*Set at the beginning of payment confirmation* | No |

Status flow:


Final
Error
Verified
Verify
Draft
Approved
Auth Type
Authorization
InProcess
Executed
Refunded
PreAuthorization
Waiting
PostAuthorization
Canceled