e-Commerce Vostok

v1.0.1

Content

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.

Since we provide Hosted Checkout page, you do not need additional integrations with Google – the GPay button will be available to all customers. If you want to disable GPay payment method - please contact our support ec@bankvostok.com.ua

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.

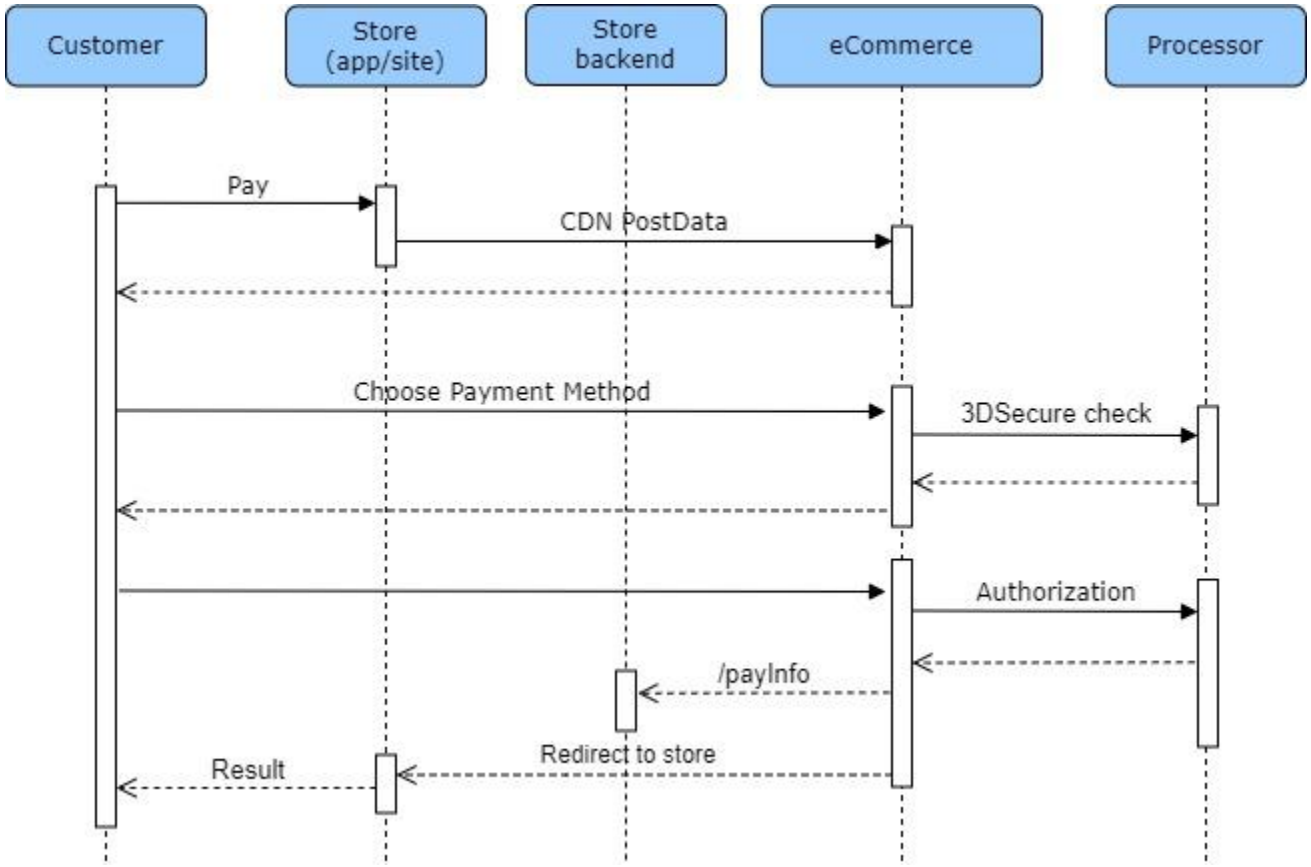
This payment method is in development and will soon be available.

1. Payment process

1.1. Payment process flow

1.1.1. Regular payment.

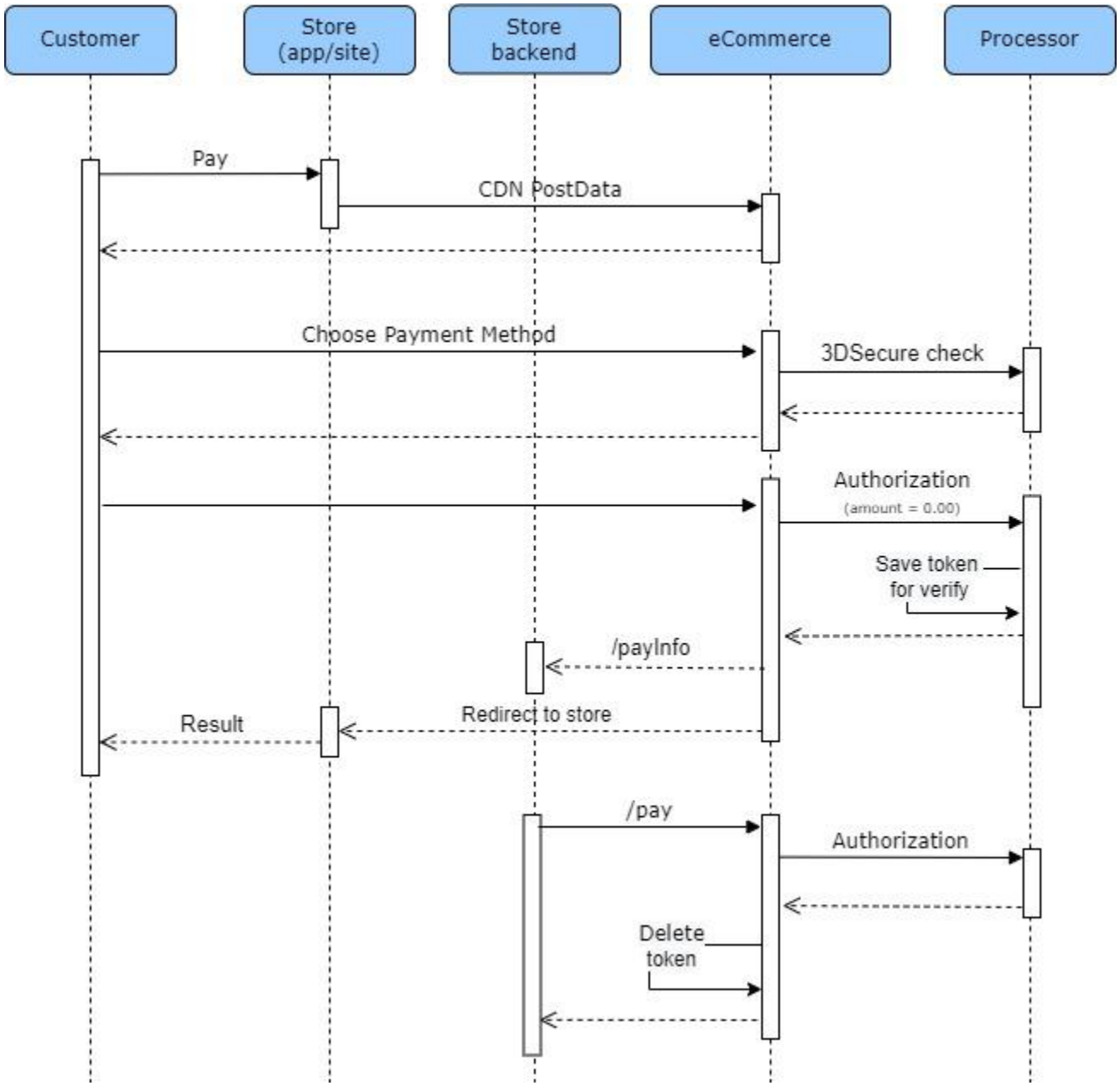
Debiting funds from the client's card during payment processing.



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.

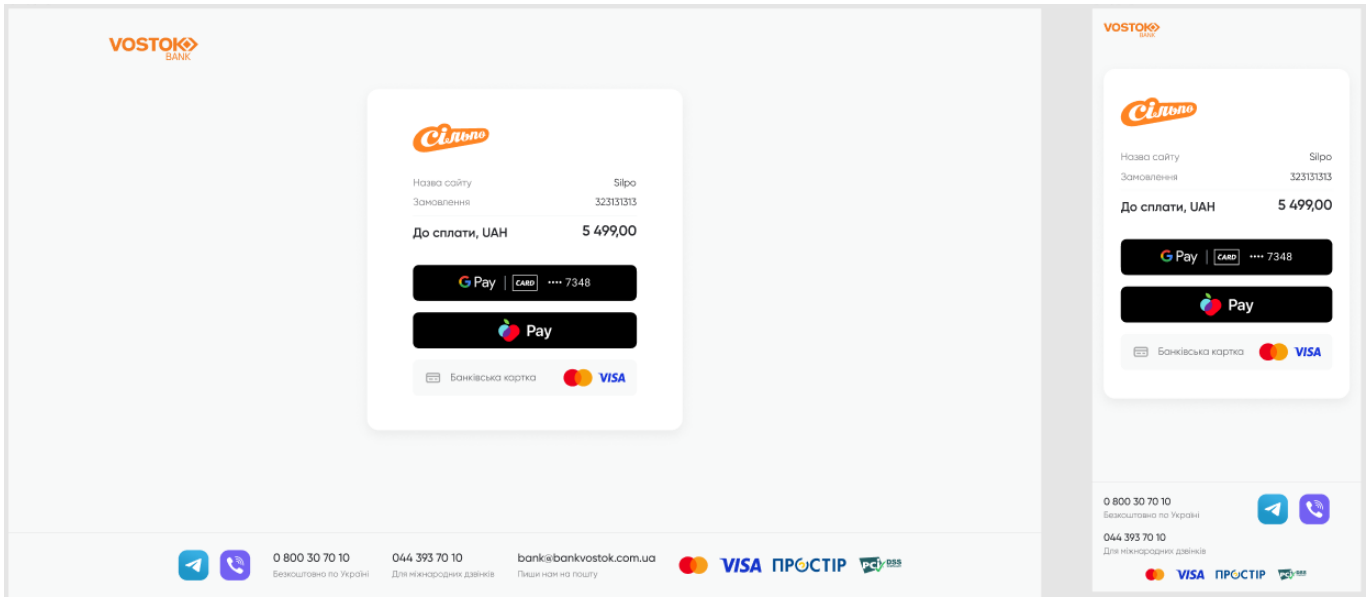


1.2. Process description

Regular payment (authType = Authorization)	2-step payment (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. 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. 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. 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: 8.1. If the payment is successful, redirects to SuccessRedirectUrl. 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 and mobile versions.



1.3. 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

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}.

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.

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, hashAlgorithm, RSASignaturePadding.Pkcs1);
return Convert.ToBase64String(signature);
```

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.

Field description:

Field	Data type	Description	Validation
phoneNumber	string	Phone number	Only Ukrainian numbers are allowed (for example, 0981234567, +38 098 1234567, (098) 123-45-67).
amount*	number	Order amount	0-9, delimiter “.” (point), value >=0
description	string	Order name	Max. 255 characters
partnerOrderId*	string	Unique order number in the Store	0-9, a-Z
merchantId*	Int32	Unique eCom merchant identifier	
authType*	Int32	Authorization type	Possible options: 1 - Authorization 2 - PreAuthorization 4 - Verify
successRedirectUrl	string	Store’s URL to which the customer will to be redirected after successful payment	0-9, a-Z, /: starts with https://
failureRedirectUrl	string	Store’s URL to which the customer will to be redirected after payment error	0-9, a-Z, /: starts with https://
callbackUrl	string	Store’s callback URL to which will be send payment result	0-9, a-Z, /: starts with https://
cultureName	string	Response localization	Possible options: uk-UA (default), en-US
signature*	string	Signature	Base 64 format

Field	Data type	Description	Validation
keyHash*	string	Private key hash	0-9, A-Z
customParameters	string	Additional parameters	Key-value parameters F.e., filialId = "QR1234"

* - required fields.

authType possible 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. 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.

2.3. eCom API methods

URL's for sending requests:

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

2.3.1. getOrder

Type: *POST*.

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

Input parameters:

Parameter	Data type	Description	Validation
partnerOrderId*	string	Unique Order ID in the merchant system	0-9, a-Z
merchantId*	Int32	Unique e-Com merchant identifier	0-9
cultureName*	string	Response localization	Currently only available uk-UA

* - required fields.

Output parameters:

Parameter	Data type	Description
errorCode*	Int32	Error code
errorMessage	string	Error text
phoneNumber	Int64	Customer's phone number
amount*	number	Order amount
currency*	string	Order currency
description	string	Order name
orderId*	Int32	Unique eCom order identifier

Parameter	Data type	Description
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 4-4 format (1111********1111)

* - required fields.

Error Codes:

Code	Description
200	OK
400	Bad Request
500	Error

2.3.2. 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	Data type	Description	Validation
partnerOrderId*	string	Unique Order ID in the merchant system	0-9, a-Z
merchantId*	Int32	Unique eCom merchant identifier	0-9
amount*	number	Amount of payment	0-9, delimiter "." (point), value >=0 For PostAuthorization, it cannot exceed the original order amount.
cardToken	string	Disposable card token	0-9, a-Z The token is valid only for the current order and is disposed of after successful payment.
cultureName*	string	Response localization	Currently only available uk-UA

* - required fields.

Output parameters:

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

Error Codes:

Code	Description
200	OK
400	Bad Request
500	Error

2.3.3. 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	Data type	Description	Validation
partnerOrderId*	string	Unique Order ID in the merchant system	0-9, a-Z
merchantid*	Int32	Unique eCom merchant identifier	0-9
amount	number	Refund amount (for partial refunds)	0-9, delimiter "." (point) If not specified, a refund is made for the purchase amount
cultureName*	string	Response localization	Currently only available uk-UA

* - required fields.

Output parameters:

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

Error Codes:

Code	Description
200	OK
400	Bad Request
500	Error

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 4-4 format (1111********1111)
paymentMethod	string	Payment method (card, bvrpay, gpay, apay)

2.4. Order statuses

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

Status flow: