Poplapay Server API (2.0.0)

Download OpenAPI specification: Download

This is a description of the public API that Poplapay provides for merchants and other integrators.

Endpoints and Versioning

There are separate production and development environments, with separate API endpoints. Production environment uses actual production credit cards and transactions involve real money. Development environment uses bank issued test cards or certification test cards and does not involve real money. Production cards must never be used with development environment and development cards must never be used with production environment.

- Production endpoint: https://api.poplatek.com/api/v2/
- Development endpoint: https://api.sandbox.poplatek.com/api/v2/

The API versioning follows Semantic Versioning 2.0.0 specification. Versions are specified as MAJOR.MINOR.PATCH, where:

- MAJOR is incremented when incompatible API changes are made
- MINOR is incremented when new functionality is added in a backwards-compatible manner

- PATCH is incremented when backward compatible bug fixes are made

All methods include the major version in the method path as `/api/MAJOR`. Each API major version is supported for backwards compatibility for **12 months** from the publication of new major version.

Authentication

All API requests require authentication. Currently HTTP basic authentication is used for all requests. Send email to developers@poplapay.com to request a development access key for authentication.

basic

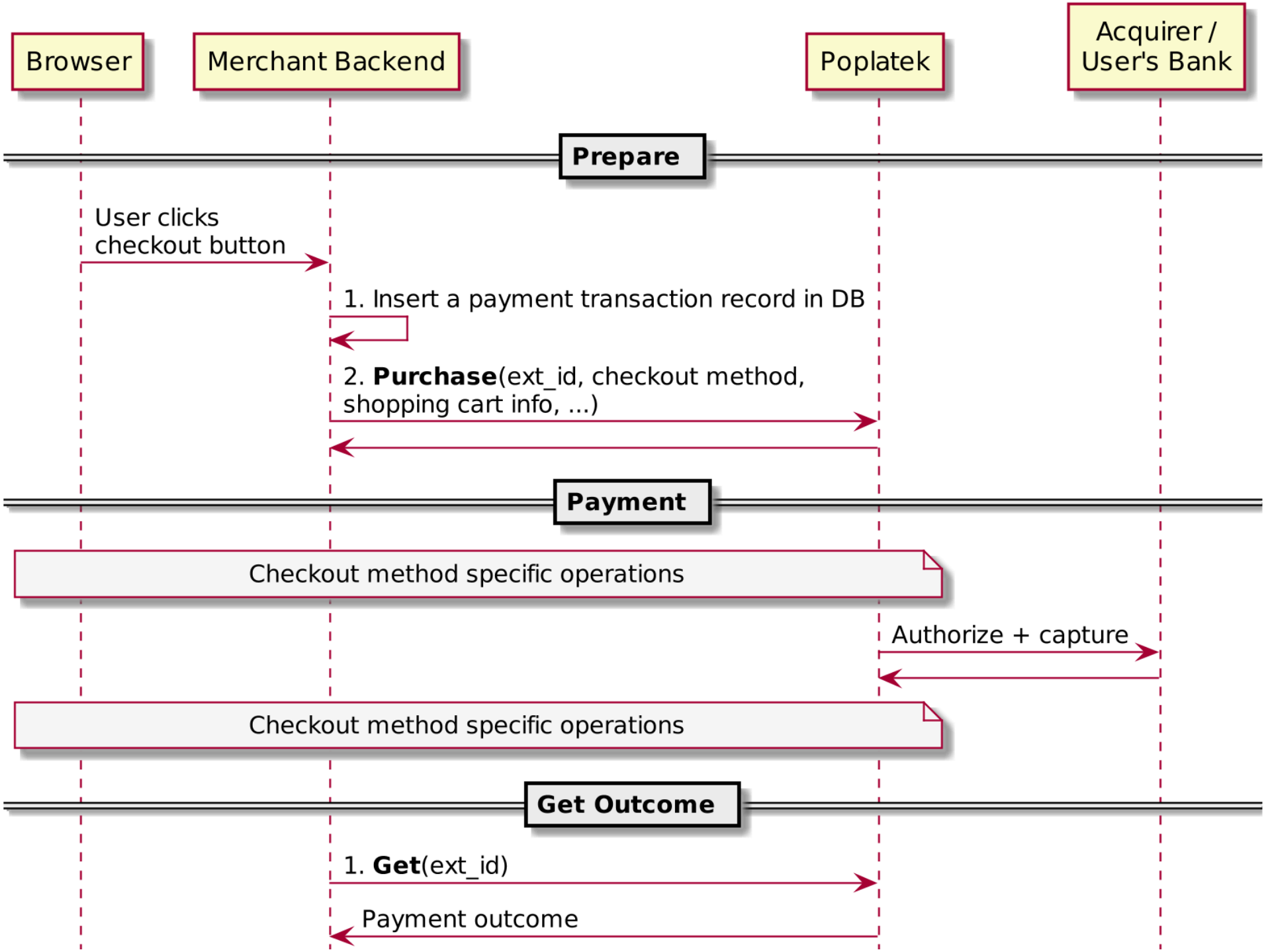
Security Scheme Type	HTTP
HTTP Authorization Scheme	basic

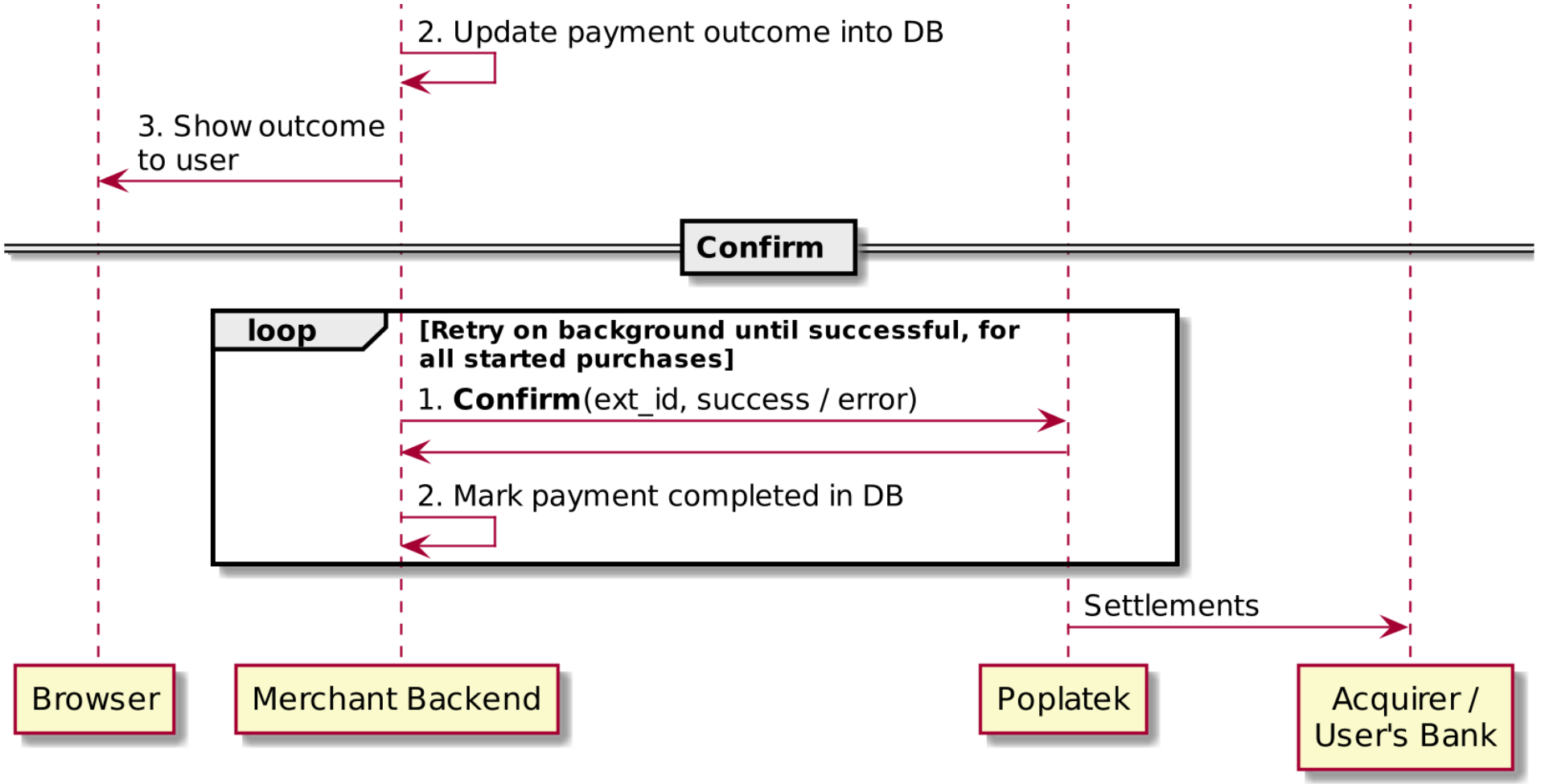
eCommerce Payments

The Payment API supports eCommerce payments using payment cards. Implementation requires implementing background processing and storage of payment transaction records at the merchant backend. The API supports deep integration of multiple checkout methods into the merchant site both at shopping cart and checkout phases of the payment process.

Purchase overview

The purchase flow always involves four stages as shown below.


Browser
Merchant Backend
Poplatek
Acquirer / User's Bank
Prepare
User clicks checkout button
1. Insert a payment transaction record in DB
2. Purchase(ext_id, checkout method, shopping cart info, ...)
Payment
Checkout method specific operations
Authorize + capture
Checkout method specific operations
Get Outcome
1. Get(ext_id)
Payment outcome



Prepare

All payments are initiated the same way. First insert a payment transaction record into merchant backend database. Then call **Purchase**, passing among other things a unique ID of the record in the merchant database.

1. Insert a new payment transaction record into merchant database. The record must contain an ID that will never be used again, eg. a table primary key or an UUID. This identifies the payment transaction, and is called `ext_id`.
2. Call **Purchase** endpoint, with parameters including:
    - `ext_id` from the previous step

- Selected checkout method, or payment form
- Shopping cart information

Response contains checkout method specific information.

Payment

The payment is authorized by user's bank during this stage. Unlike other stages, this one is not performed by calling a server-to-server API endpoint. Details of this stage are different for each checkout method.

Often payment is performed by redirecting user to a URL to enter payment details, and redirected back to merchant's server.

Alternatively, there may need to be checkout method specific JavaScript code on the merchant page.

Get Outcome

After payment stage has been completed, the merchant backend must read payment transaction outcome (success or failure) by calling **Get** endpoint. Merchant must then update its own database, and finally show the purchase result to user.

1. Call **Get** endpoint, passing `ext_id`. Response indicates among other things if the payment transaction was successful.
2. Store the outcome information into the payment transaction record in the merchant database.

3. Show the outcome to user.

Notice that the payment may have been successfully authorized in the Payment stage even if this stage isn't reached, eg. because user fails to redirect back to the merchant's server. See "Transactional Reliability" for more information.

Confirm

Confirm must be called for all transactions for which a **Purchase** call has been completed or attempted, regardless of the payment outcome. This stage must be performed on the background, because the **Confirm** call must be retried until it succeeds.

1. Call **Confirm** endpoint, passing `ext_id`. Even if **Get** indicates the payment transaction was successful, it's acceptable to confirm it as having failed.
2. Mark the payment transaction record as confirmed in the merchant database.

If **Confirm** is not called, the transaction will not be cleared, and manual processing may be required. Merchant will not receive payment even if authorization was successful.

Checkout methods

Payment API supports multiple different checkout methods, given in **Purchase** request field `checkout_method`. In all the methods, the "Prepare", "Get Outcome" and "Confirm" stages are similar. The largest differences are in user interface. Generally, the deeper the checkout method integrates with the merchant platform (eg. a web site or mobile app), the larger the implementation effort and the potential for streamlined user experience.

PAYMENT_FORM

In the `PAYMENT_FORM` checkout method merchant redirects to a page where user enters card information. On completion user returns to `return_url`. There are no branding restrictions concerning the checkout button or otherwise.

1. When user clicks on the checkout button, create a record in merchant database and then call **Purchase**, as described in "Prepare" section above.
2. Redirect user to the payment form URL in **Purchase** response field `payment_form.redirect_url`.
3. When user is redirected to `return_url` given in the **Purchase** call, the "Payment" stage is complete
4. Perform the steps described in the "Get Outcome" section above.
5. Interactive flow is now completed. Ensure that **Confirm** is eventually called by merchant's server as described in the "Confirm" section above.

Cancelling a purchase

A purchase can be cancelled before it has been settled with the acquirer, by calling **Cancel**. This does not create a new payment transaction, but modifies the purchase transaction.

Refunding

A successful purchase can be refunded for 40 days by calling **Refund**. This creates a new payment transaction, which will be referenced by the purchase transaction. There can be multiple partial refunds for a single purchase transaction.

Reporting API

Settlement

Returns settlement batches and their transactions for given day. Failed transactions are not present in any batch. Response may contain additional fields not specified in this document. Such fields should be ignored.

Obs! Notice that while other endpoints of the server api are at /api/v2/ the settlement report is at /api/v1/.

QUERY PARAMETERS

`date` required	string Example: `date=2014-05-20Z` Creation date of the settlement batches.

Responses

> **200** Success

> **500** Failed

GET `/api/v1/report/settlement`

Response samples

200 **500**

Content type

application/json

Copy Expand all Collapse all

```
[
 - {
      "creation_time": "2024-11-06T07:23:05.681Z",
      "currency": 978,
      "currency_alpha": "EUR",
      "feedback_status": "ACCEPTED",
      "merchant_id": 12345,
      "merchant_number": "2100112307",
      "purchases_amount": 83750,
      "purchases_count": 51,
      "reference_number": null,
      "refunds_amount": 4995,
      "refunds_count": 3,
      "sales_location_id": 12345,
      "settlement_batch_id": "517",
      "state": "DELIVERED",
    + "transactions": [ … ]
   }
]
```

Payment API

Cancel

Cancels a purchase transaction completely. The customer will not be charged for the transaction after cancellation, and any reserved funds will be freed. Cancellation for a partial amount is not supported.

Cancellation is not possible for transactions which have already been settled with the acquirer. Such transactions will have to be refunded instead.

As opposed to **Refund** which creates a new payment transaction, **Cancel** modifies the existing purchase transaction by setting `status_code` to `USER_CANCELLED`.

Returns the cancelled payment transaction the same way as **Get**.

If the payment transaction is already cancelled, its information will be returned unchanged. Therefore, this call is idempotent, and can be safely retried.

AUTHORIZATIONS: basic

REQUEST BODY SCHEMA: application/json

`ext_id`
string
Payment transaction identifier given by the calling system. Handled as an opaque string, and used together with `ext_scope` to uniquely identify the transaction.

`ext_scope`
string
Identifier scope for `ext_id`. Usually automatically derived from caller's authentication. The pair of `ext_scope` and `ext_id` is guaranteed to uniquely identify a single payment transaction.

`unique_id`
string
Payment transaction identifier assigned by Poplapay. Unique across all `ext_scope` values, so may be easier to use if `ext_scope` can't be automatically derived from caller's authentication. When passed, possible `ext_id` and `ext_scope` parameters will be ignored.

`reason_code`
required

string^[A-Z_]+$
Reason for the cancellation.

- `USER_CANCELLED`: Customer cancelled the purchase by his own action.
- `TIMEOUT`: Some timeout was triggered during purchase processing, and the purchase was cancelled.
- `MERCHANT_CANCELLED`: Merchant decided to cancel the purchase.
- `UNKNOWN`: Other cancellation reason. `reason_description` should give more information.

Other codes can be used, but should be agreed with Poplapay.

`reason_description`

string
A human readable description of the cancellation reason. This will not be presented to the user, but will be shown in logs, and administrative interfaces, and refund transaction's information.

Responses

> **200** Success

> **500** Failed

POST `/api/v2/payment/cancel`

Request samples

Payload

Content type

application/json

Copy Expand all Collapse all


```
{
   "ext_id": "a526ceca-565e-493e-aad6-a4912b5453c3",
   "ext_scope": "web123",
   "unique_id": "2a:1087143940",
   "reason_code": "USER_CANCELLED",
   "reason_description": "string"
}
```


Response samples

200 500

Content type

application/json

Copy Expand all Collapse all


```
{
   "ext_id": "a526ceca-565e-493e-aad6-a4912b5453c3",
   "ext_scope": "web123",
   "unique_id": "2a:1087143940",
   "status_code": "SUCCESS",
   "status_description": "Transaction successful.",
   "status_details": "string",
   "state": "PREPARE",
   "transaction_type": "PURCHASE",
   "timestamp": "2016-10-12T11:06:19.241Z",
```

```
  "confirm_timestamp": "2016-10-12T11:06:27.090Z",
  "merchant_id": 9413,
  "settlement_contract_id": 1426,
  "sales_location_id": 38244,
  "terminal_id": 10353,
  "hardware_id": "a0f6fd72e785",
  "amount": 1200,
  "cashback_amount": 0,
  "refundable_amount": 1200,
  "currency": 978,
  "cardholder_verification": "PIN",
  "entry_mode": "chip",
  "card_number_customer": "************0035",
  "card_number_merchant": "411122******0035",
  "card_scheme": "VI",
  "card_country": 246,
  "card_currency": 978,
  "acquirer_timestamp": "161012140619",
  "acquirer_reference": "161012123253",
  "order_id": "5e98ebdeb3a17b48012400746a18ca9a",
  "order_description": "5e98ebdeb3a17b48012400746a18ca9a",
  "browser_ip_country": 246,
  "authorized": true,
  "authorized_amount": 1200,
  "authorization_code": "string",
  "id_check_performed": true,
- "external_data": {
     "name": "John Doe",
   + "shift": { … }
  },
  "referring_transactions": "2a:1087143945"
```

```
}
```

Refund

Refunds an existing purchase transaction fully or partially. It's possible to refund a transaction partially multiple times, as long as the sum of refunds does not exceed `amount` of the original transaction.

Refund is usually possible within 70 days of purchase, but this may vary.

The original purchase transaction that this is a refund for is identified by parameters `original_ext_id`, `original_ext_scope` and `original_unique_id`, which can be given in similar combinations as **Get**'s `ext_id`, `ext_scope` and `unique_id`.

The original transaction must be successful (`status_code=SUCCESS`) and it must be confirmed (`state=CLOSED`). If these conditions don't hold, the transaction won't be settled with the acquirer and so refunding would make no sense.

This request will create a new payment transaction with given `ext_id` and `transaction_type=REFUND`. The transaction will need to be confirmed with the **Confirm** call.

The `unique_id` that is automatically assigned to this refund transaction will be added to `referring_transactions` field of the original purchase transaction.

Returns the created refund transaction the same way as **Get**.

If a refund transaction with the same `ext_id` and `ext_scope` already exists, its information will be returned unchanged, and no changes to the original purchase transaction are made. Therefore, this call is idempotent, and can be safely retried.

AUTHORIZATIONS: basic

REQUEST BODY SCHEMA: application/json

`ext_id` string

required

Payment transaction identifier given by the calling system. Handled as an opaque string, and used together with `ext_scope` to uniquely identify the transaction.

`ext_scope`

string

Identifier scope for `ext_id`. Usually automatically derived from caller's authentication. The pair of `ext_scope` and `ext_id` is guaranteed to uniquely identify a single payment transaction.

`original_ext_id`

string

Original purchase transaction's `ext_id`.

`original_ext_scope`

string

Original purchase transaction's `ext_scope`. If not given, defaults to this request's `ext_scope`.

`original_unique_id`

string

Original purchase transaction's `unique_id`.

`amount`
required

integer `>= 0`

The amount to be refunded. This may be equal to or smaller than the current `refundable_amount` for the purchase transaction. For exact format, see field of the same name in **Get** response.

`currency`
required

integer `[ 0 .. 999 ]`

Currency that the `amount` is in. For exact format, see field of the same name in **Get** response.

`reason_code`
required

string^[A-Z_]+$

Reason for the refund.

- `MERCHANT_REFUND`: Merchant decided to refund part or all of the purchase. The reason for this may be product returns, inability to deliver, or some other reason.
- `UNKNOWN`: Other refund reason. `reason_description` should give more information.

Other codes can be used, but should be agreed with Poplapay.

`reason_description`

string

	A human readable description of the refund reason. This will not be presented to the user, but will be shown in logs and administrative interfaces, and refund transaction's information.
`transaction_time`	string This field is required if the relevant feature is enabled for the account. Otherwise this field is ignored. Value is the transaction timestamp in local time as `YYMMDDHHMMSS`. Must be within 5 minutes of current time.
`reference_number`	string This field is required if the relevant feature is enabled for the account. Otherwise this field is ignored. The first six digits must match the the same digits of transaction time. The last six digits must be between 100000-899999. Hence, the format is `YYMMDDNNNNNN`. Must be unique among all transactions of the payment terminal contract.

Responses

> **200** Success

> **500** Failed

POST `/api/v2/payment/refund`

Request samples

Payload

Content type

application/json

Copy Expand all Collapse all


```
{
    "ext_id": "a526ceca-565e-493e-aad6-a4912b5453c3",
    "ext_scope": "web123",
    "original_ext_id": "a526ceca-565e-493e-aad6-a4912b5453c3",
    "original_ext_scope": "web123",
    "original_unique_id": "2a:1087143940",
    "amount": 1200,
    "currency": 978,
    "reason_code": "MERCHANT_REFUND",
    "reason_description": "string",
    "transaction_time": "161012140619",
    "reference_number": "161012123253"
}
```


Response samples

200 500

Content type

application/json

Copy Expand all Collapse all


```
{
    "ext_id": "a526ceca-565e-493e-aad6-a4912b5453c3",
    "ext_scope": "web123",
    "unique_id": "2a:1087143940",
    "status_code": "SUCCESS",
    "status_description": "Transaction successful.",
    "status_details": "string",
    "state": "PREPARE",
```

```
"transaction_type": "PURCHASE",
"timestamp": "2016-10-12T11:06:19.241Z",
"confirm_timestamp": "2016-10-12T11:06:27.090Z",
"merchant_id": 9413,
"settlement_contract_id": 1426,
"sales_location_id": 38244,
"terminal_id": 10353,
"hardware_id": "a0f6fd72e785",
"amount": 1200,
"cashback_amount": 0,
"refundable_amount": 1200,
"currency": 978,
"cardholder_verification": "PIN",
"entry_mode": "chip",
"card_number_customer": "************0035",
"card_number_merchant": "411122******0035",
"card_scheme": "VI",
"card_country": 246,
"card_currency": 978,
"acquirer_timestamp": "161012140619",
"acquirer_reference": "161012123253",
"order_id": "5e98ebdeb3a17b48012400746a18ca9a",
"order_description": "5e98ebdeb3a17b48012400746a18ca9a",
"browser_ip_country": 246,
"authorized": true,
"authorized_amount": 1200,
"authorization_code": "string",
"id_check_performed": true,
```

```
 - "external_data": {
       "name": "John Doe",
    + "shift": { … }
   },
   "referring_transactions": "2a:1087143945"
}
```

Get

Get full information about a payment transaction.

A transaction is considered successful if `status_code` field has value `SUCCESS`. If `state` field has value other than `CLOSED`, the payment flow is not yet completed, and `status_code` may still change.

changed eg. if a **Confirm** call marks the transaction has failed.

AUTHORIZATIONS: basic

REQUEST BODY SCHEMA: application/json

`ext_id` — string
Payment transaction identifier given by the calling system. Handled as an opaque string, and used together with `ext_scope` to uniquely identify the transaction.

`ext_scope` — string
Identifier scope for `ext_id`. Usually automatically derived from caller's authentication. The pair of `ext_scope` and `ext_id` is guaranteed to uniquely identify a single payment transaction.

`unique_id`

string

Payment transaction identifier assigned by Poplapay. Unique across all `ext_scope` values, so may be easier to use if `ext_scope` can't be automatically derived from caller's authentication. When passed, possible `ext_id` and `ext_scope` parameters will be ignored.

`terminal_id`

integer

Poplapay-assigned identifier for the particular payment site. A single merchant may have several differently configured payment sites, which are identified by their unique `terminal_id` values. The field name `terminal_id` refers to a similarily named field in acquirer protocols, and is shared with physical payment terminal transactions.

The pair of `terminal_id` and `reference_number` is normally enough to uniquely identify a single payment transaction.

`reference_number`

string

Twelve-digit reference number of the financial transaction.

The pair of `terminal_id` and `reference_number` is normally enough to uniquely identify a single payment transaction.

Responses

> **200** Success

> **500** Failed

POST `/api/v2/payment/get`

Request samples

Payload

Content type

application/json

Copy Expand all Collapse all


```
{
   "ext_id": "a526ceca-565e-493e-aad6-a4912b5453c3",
   "ext_scope": "web123",
   "unique_id": "2a:1087143940",
   "terminal_id": 15354,
   "reference_number": "150304000133"
}
```


Response samples

200 **500**

Content type

application/json

Copy Expand all Collapse all


```
{
   "ext_id": "a526ceca-565e-493e-aad6-a4912b5453c3",
```

```
"ext_scope": "web123",
"unique_id": "2a:1087143940",
"status_code": "SUCCESS",
"status_description": "Transaction successful.",
"status_details": "string",
"state": "PREPARE",
"transaction_type": "PURCHASE",
"timestamp": "2016-10-12T11:06:19.241Z",
"confirm_timestamp": "2016-10-12T11:06:27.090Z",
"merchant_id": 9413,
"settlement_contract_id": 1426,
"sales_location_id": 38244,
"terminal_id": 10353,
"hardware_id": "a0f6fd72e785",
"amount": 1200,
"cashback_amount": 0,
"refundable_amount": 1200,
"currency": 978,
"cardholder_verification": "PIN",
"entry_mode": "chip",
"card_number_customer": "************0035",
"card_number_merchant": "411122******0035",
"card_scheme": "VI",
"card_country": 246,
"card_currency": 978,
"acquirer_timestamp": "161012140619",
"acquirer_reference": "161012123253",
"order_id": "5e98ebdeb3a17b48012400746a18ca9a",
"order_description": "5e98ebdeb3a17b48012400746a18ca9a",
"browser_ip_country": 246,
"authorized": true,
```

```
    "authorized_amount": 1200,
    "authorization_code": "string",
    "id_check_performed": true,
  - "external_data": {
        "name": "John Doe",
      + "shift": { … }
    },
    "referring_transactions": "2a:1087143945"
}
```

Confirm

Confirms a purchase or refund transaction as successful or failed.

If purchase processing has been successful in merchant site (merchant's customer will receive the purchased product), confirm the transaction as successful by passing value `SUCCESS` in `result_code`. Otherwise use any other value, which will be considered an error code. After this call, transaction's `state` will be `CLOSED`.

If **Get** response indicates the transaction was successful (`status_code=SUCCESS`) it can be confirmed as either successful or failed. Otherwise it must be confirmed as failed. After being confirmed as failed, transaction's `status_code` will have a value other than `SUCCESS`.

A call will be successful (HTTP status 200) even if the specified transaction does not exist. This enables calling **Confirm** for all `ext_id`'s for which **Purchase** call has been attempted.

Transaction's `status_code` may still change after this call has completed, for example as a result of a **Cancel** call.

This endpoint shouldn't be called as part of user interaction, but from a background thread, because a failed call always requires retrying until successful. See "Transactional Reliability" for more discussion.

AUTHORIZATIONS: basic

REQUEST BODY SCHEMA: application/json

`ext_id`

string

Payment transaction identifier given by the calling system. Handled as an opaque string, and used together with `ext_scope` to uniquely identify the transaction.

`ext_scope`

string

Identifier scope for `ext_id`. Usually automatically derived from caller's authentication. The pair of `ext_scope` and `ext_id` is guaranteed to uniquely identify a single payment transaction.

`result_code`
required

string^[A-Z_]+$

Pass `SUCCESS` if **Get** response indicates the payment transaction was successful (`status_code=SUCCESS`), and no errors have occurred in merchant site either. Otherwise pass any other value, which will be considered an error code.

`result_description`

string

Usually only used if `result_code` is not `SUCCESS`. An English human readable error description to be used for debugging. The value will be visible in **Get** response field `status_description`, if that field was formerly empty.

`result_details`

string

Usually only used if `result_code` is not `SUCCESS`. A stack traceback or other verbose context information about the error. The value will be visible in **Get** response field `status_details`, if that field was formerly empty.

Responses

— **200** Success

› **500** Failed

POST `/api/v2/payment/confirm`

Request samples

Payload

Content type

application/json

Copy Expand all Collapse all


```
{
    "ext_id": "a526ceca-565e-493e-aad6-a4912b5453c3",
    "ext_scope": "web123",
    "result_code": "DB_ERROR",
    "result_description": "Database update failed: connection refused",
    "result_details": "Traceback (most recent call last): ..."
}
```


Response samples

500

Content type

application/json

Copy Expand all Collapse all

```
{
    "error_code": "NOT_FOUND",
    "error_description": "Missing required parameter: foo",
    "error_details": "Traceback (most recent call last): ..."
}
```

Merchant API

Get hardware

Get full information about hardware, and basic information about the terminal contract, sales location, and merchant the hardware may be attached to.

AUTHORIZATIONS: basic

REQUEST BODY SCHEMA: application/json

`hardware_id`
required

string
Unique ID (sometimes called "MAC") of the hardware.

Responses

> **200** Success

> **500** Failed

POST `/api/v2/merchant/hardware/get`

Request samples

Payload

Content type

application/json

Copy Expand all Collapse all

```
{
    "hardware_id": "0001234abcd1"
}
```

Response samples

200 **500**

Content type

application/json

Copy Expand all Collapse all


```
{
    "hardware_id": "0001234abcd1",
    "hardware_model": "YOMANI ML",
    "ownership": "OWNED",
  - "terminal_contract": {
        "terminal_contract_id": 123456,
        "terminal_contract_type": "spm20-poplatek-offline",
        "display_name": "POS 3",
      + "sales_location": { … }
    }
}
```


Attach hardware

Attach hardware (a physical terminal) to an existing terminal contract. This operation has no effect on billing, which is based on active terminal contracts. Multiple hardwares should be attached to a terminal contract only temporarily, eg. when replacing a broken hardware. Successful response is an empty object. If the hardware is already attached to the terminal contract, a success response is returned. This way retrying a timed out request is straightforward.

AUTHORIZATIONS: basic

REQUEST BODY SCHEMA: application/json

`terminal_contract_id` required	integer ID of the terminal contract to attach the hardware to.

`hardware_id` required	string ID (sometimes called "MAC") of the hardware to attach to the terminal contract. ID is not case sensitive.

Responses

— **200** Success

› **500** Failed

POST `/api/v2/merchant/terminal_contract/attach_hardware`

Request samples

Payload

Content type

application/json

Copy Expand all Collapse all


```
{
   "terminal_contract_id": 1234,
   "hardware_id": "0001234abcd1"
}
```

Response samples

500

Content type

application/json

Copy Expand all Collapse all


```
{
    "error_code": "INVALID_HARDWARE",
    "error_description": "Missing required parameter: foo",
    "error_details": "Traceback (most recent call last): ..."
}
```

Detach hardware

Detach hardware (a physical terminal) from a terminal contract. This operation has no effect on billing, which is based on active terminal contracts. Hardware is typically detached when the terminal contract period has ended, or when the hardware is broken and replacement hardware is attached instead. Successful response is an empty object. If the hardware is already not attached to the terminal contract, a success response is returned. This way retrying a timed out request is straightforward.

AUTHORIZATIONS: basic

REQUEST BODY SCHEMA: application/json

`terminal_contract_id` required	integer ID of the terminal contract to detach the hardware from.

hardware_id required	string ID (sometimes called "MAC") of the hardware to detach from the terminal contract. ID is not case sensitive.

Responses

— **200** Success

› **500** Failed

POST /api/v2/merchant/terminal_contract/detach_hardware

Request samples

Payload

Content type

application/json

Copy Expand all Collapse all


```
{
   "terminal_contract_id": 1234,
   "hardware_id": "0001234abcd1"
}
```

Response samples

500

Content type

application/json

Copy Expand all Collapse all


```
{
    "error_code": "INVALID_TERMINAL_CONTRACT",
    "error_description": "Missing required parameter: foo",
    "error_details": "Traceback (most recent call last): ..."
}
```


---

eCommerce API

This API covers eCommerce purchase flow. Refunds, cancellations, and getting purchase (transaction) status is performed using Payment API.

Store card

Prepares a new transaction identified by `ext_id` for processing. The `transaction_type` field will have value `ACCOUNT_CHECK`.

If a store card transaction with the same `ext_id` and `ext_scope` already exists, its information will be returned unchanged. Therefore, this call is idempotent, and can be safely retried.

The transaction will need to be confirmed with the Confirm call.

AUTHORIZATIONS: basic

REQUEST BODY SCHEMA: application/json

ext_id
required
string
Transaction identifier given by the calling system. This will be handled as an opaque string and is used to uniquely identify the transaction. This should be allocated and stored in a database before sending the request so that transactional reliability can be achieved. See section "Transactional Reliability".

terminal_id
required
integer
Poplapay-assigned identifier for the particular payment site. A single merchant may have several differently configured payment sites, which are identified by their unique `terminal_id` values. The field name `terminal_id` refers to a similarily named field in acquirer protocols, and is shared with physical payment terminal transactions.

currency
required
integer `[ 0 .. 999 ]`
Currency code of the transaction according to ISO 4217 numeric, e.g. 978 for euro.

origin_url
required
string
The URL of the page from where the checkout action will be invoked, either by navigating to an URL or some JavaScript based checkout method.

return_url
required
string
The URL of the page to which user will be redirected after payment has been completed. The same url will be used for both success and failure cases.

language
required
string^[a-z]{2}$
Default: `"en"`

The currently used interface language by the customer, as a two letter code specified by ISO 639-1, e.g. `fi` for Finnish.

`allowed_card_schemes`

Array of strings

Items Enum: `"VI"` `"MC"` `"AX"` `"DC"` `"JC"` `"UP"`

A list of one or more card schemes that the purchase is permitted to be completed with. If this field is not used, all schemes are allowed. This list will be further restricted by the settlement contracts the merchant has. List items are as following:

- `VI` : Visa and Visa Electron
- `MC` : MasterCard and Maestro
- `AX` : American Express
- `DC` : Diners Club
- `JC` : Japan Credit Bureau
- `UP` : China UnionPay

`order_id`

string

ID of the order that this purchase transaction is for. There may be multiple purchase transactions per order, usually because some of the purchases have not succeed or are partial. This is not shown in any UI, but will be visible in transaction's information so can be useful in reporting or debugging.

`order_description`

string

Freeform order description. Will be shown to the customer in eCommerce payment form, and will be retained in logs but not processed programmatically. Required if `checkout_method` is `PAYMENT_FORM` or `STORE_CARD`.

`recurring_payment`

boolean

Default: `false`

To create recurring payment of transaction. This is just indicator for Nets. Merchant backend still needs to invoke payments in wanted cycle with `checkout_method` as `STORED_CARD`.

Responses

> **200** Success

> **500** Failed

POST `/api/v2/card/store`

Request samples

Payload

Content type

application/json

Copy Expand all Collapse all


```
{
   "ext_id": "c2c8efe7-8b54-430b-b843-79cef6529948",
   "terminal_id": 15354,
   "currency": 978,
   "origin_url": "https://example.com/merchant/page/checkout",
   "return_url": "https://example.com/merchant/page/checkoutcomplete",
   "language": "fi",
 - "allowed_card_schemes": [
      "VI",
      "MC",
      "AX"
   ],
   "order_id": "5e98ebdeb3a17b48012400746a18ca9a",
```

```
    "order_description": "5e98ebdeb3a17b48012400746a18ca9a",
    "recurring_payment": false
}
```


Response samples

200 | 500

Content type

application/json

Copy Expand all Collapse all


```
{
 - "store_card_form": {
      "redirect_url": "https://example.com/payment_form/page/"
   },
   "ext_id": "a526ceca-565e-493e-aad6-a4912b5453c3",
   "ext_scope": "web123",
   "unique_id": "2a:1087143940",
   "status_code": "SUCCESS",
   "status_description": "Transaction successful.",
   "status_details": "string",
   "state": "PREPARE",
   "transaction_type": "PURCHASE",
   "timestamp": "2016-10-12T11:06:19.241Z",
   "confirm_timestamp": "2016-10-12T11:06:27.090Z",
   "merchant_id": 9413,
   "settlement_contract_id": 1426,
   "sales_location_id": 38244,
   "terminal_id": 10353,
   "hardware_id": "a0f6fd72e785",
```

```
  "amount": 1200,
  "cashback_amount": 0,
  "refundable_amount": 1200,
  "currency": 978,
  "cardholder_verification": "PIN",
  "entry_mode": "chip",
  "card_number_customer": "************0035",
  "card_number_merchant": "411122******0035",
  "card_scheme": "VI",
  "card_country": 246,
  "card_currency": 978,
  "acquirer_timestamp": "161012140619",
  "acquirer_reference": "161012123253",
  "order_id": "5e98ebdeb3a17b48012400746a18ca9a",
  "order_description": "5e98ebdeb3a17b48012400746a18ca9a",
  "browser_ip_country": 246,
  "authorized": true,
  "authorized_amount": 1200,
  "authorization_code": "string",
  "id_check_performed": true,
- "external_data": {
     "name": "John Doe",
   + "shift": { … }
  },
  "referring_transactions": "2a:1087143945"
}
```

Remove card

Remove stored payment card with token.

AUTHORIZATIONS: basic

REQUEST BODY SCHEMA: application/json

`token`
required

string
Card token which is received when done card store or purchase with card storing option.

Responses

> **200** Success

> **500** Failed

POST `/api/v2/card/remove`

Request samples

Payload

Content type

application/json

Copy Expand all Collapse all


```
{
    "token": "c2c8efe7-8b54-430b-b843-79cef6529948"
}
```


Response samples

200 500

Content type

application/json

Copy Expand all Collapse all


```
{
    "result": "SUCCESS"
}
```

Purchase

Prepares a new payment transaction identified by `ext_id` for processing. The `transaction_type` field will have value `PURCHASE`.

If a purchase transaction with the same `ext_id` and `ext_scope` already exists, its information will be returned unchanged. Therefore, this call is idempotent, and can be safely retried.

The purchase transaction will need to be confirmed with the Confirm call.

AUTHORIZATIONS: basic

REQUEST BODY SCHEMA: application/json

ext_id
required
string
Transaction identifier given by the calling system. This will be handled as an opaque string and is used to uniquely identify the transaction. This should be allocated and stored in a database before sending the request so that transactional reliability can be achieved. See section "Transactional Reliability".

terminal_id
required
integer
Poplapay-assigned identifier for the particular payment site. A single merchant may have several differently configured payment sites, which are identified by their unique `terminal_id` values. The field name `terminal_id` refers to a similarily named field in acquirer protocols, and is shared with physical payment terminal transactions.

amount
required
integer `[ 0 .. 999999999999 ]`
Amount of the transaction expressed with implicit decimal point corresponding to the minor unit of currency as defined by ISO 4217. Eg. for 12.00 euros pass value 1200.

currency
required
integer `[ 0 .. 999 ]`
Currency code of the transaction according to ISO 4217 numeric, e.g. 978 for euro.

origin_url
required
string
The URL of the page from where the checkout action will be invoked, either by navigating to an URL or some JavaScript based checkout method.

return_url
required
string
The URL of the page to which user will be redirected after payment has been completed. The same url will be used for both success and failure cases.

`checkout_method`
required

string
Enum: `"PAYMENT_FORM"` `"STORED_CARD"` `"MOBILEPAY"`
See section "Checkout Methods" above.

`language`
required

string^[a-z]{2}$
Default: `"en"`
The currently used interface language by the customer, as a two letter code specified by ISO 639-1, e.g. `fi` for Finnish.

`allowed_card_schemes`

Array of strings
Items Enum: `"VI"` `"MC"` `"AX"` `"DC"` `"JC"` `"UP"`
A list of one or more card schemes that the purchase is permitted to be completed with. If this field is not used, all schemes are allowed. This list will be further restricted by the settlement contracts the merchant has. List items are as following:

- `VI`: Visa and Visa Electron
- `MC`: MasterCard and Maestro
- `AX`: American Express
- `DC`: Diners Club
- `JC`: Japan Credit Bureau
- `UP`: China UnionPay

`card_scheme`

string
Enum: `"VI"` `"MC"` `"AX"` `"DC"` `"JC"` `"UP"`
Card scheme the purchase is meant to be completed with, but user can use any other scheme permitted by `allowed_card_schemes`. Completed transaction will show which scheme was ultimately used for transaction.

`reject_unauthenticated`

boolean
Default: `false`
Reject purchases that cannot be authenticated via Visa 3-D Secure, MasterCard SecureCode or similar technology. Such purchases carry higher risk for the merchant, but provide an easier payment experience for the customer.

`prefer_unauthenticated`

boolean

Default: `false`

Attempt to complete payment without using Visa 3-D Secure, MasterCard SecureCode or similar technology for authentication. Such purchases carry higher risk for merchant, but provide an easier payment experience for the customer.

`order_id`

string

ID of the order that this purchase transaction is for. There may be multiple purchase transactions per order, usually because some of the purchases have not succeed or are partial. This is not shown in any UI, but will be visible in transaction's information so can be useful in reporting or debugging.

`order_description`

string

Freeform order description. Will be shown to the customer in eCommerce payment form, and will be retained in logs but not processed programmatically. Required if `checkout_method` is `PAYMENT_FORM`.

`store_card`

string

Default: `null`

Enum: `"REQUIRED"` `"OPTIONAL"` `"DISABLED"`

Store card options determines how form is displayed for user. `REQUIRED` forces user to store card. `OPTIONAL` allows user to choose wheter to store card. `DISABLED` doesn't allow user to store card at all. If not given `DISABLED` is default value.

`recurring_payment`

boolean

Default: `false`

To create recurring payment of transaction. This is just indicator for Nets. Merchant backend still needs to invoke payments in wanted cycle with `checkout_method` as `STORED_CARD`.

`shopping_cart` ›

Array of objects

List of shopping cart items. Currently not used.

Responses

> **200** Success

> **500** Failed

POST `/api/v2/payment/purchase`

Request samples

Payload

Content type

application/json

Copy Expand all Collapse all


```
{
    "ext_id": "c2c8efe7-8b54-430b-b843-79cef6529948",
    "terminal_id": 15354,
    "amount": 1200,
    "currency": 978,
    "origin_url": "https://example.com/merchant/page/checkout",
    "return_url": "https://example.com/merchant/page/checkoutcomplete",
    "checkout_method": "PAYMENT_FORM",
    "language": "fi",
```

```
  - "allowed_card_schemes": [
        "VI",
        "MC",
        "AX"
    ],
    "card_scheme": "VI",
    "reject_unauthenticated": false,
    "prefer_unauthenticated": true,
    "order_id": "5e98ebdeb3a17b48012400746a18ca9a",
    "order_description": "5e98ebdeb3a17b48012400746a18ca9a",
    "store_card": "REQUIRED",
    "recurring_payment": false,
  - "shopping_cart": [
     + { … }
    ]
}
```


Response samples

200 500

Content type

application/json

Copy Expand all Collapse all


```
{
  - "payment_form": {
        "redirect_url": "string"
    },
    "ext_id": "a526ceca-565e-493e-aad6-a4912b5453c3",
    "ext_scope": "web123",
```

```
"unique_id": "2a:1087143940",
"status_code": "SUCCESS",
"status_description": "Transaction successful.",
"status_details": "string",
"state": "PREPARE",
"transaction_type": "PURCHASE",
"timestamp": "2016-10-12T11:06:19.241Z",
"confirm_timestamp": "2016-10-12T11:06:27.090Z",
"merchant_id": 9413,
"settlement_contract_id": 1426,
"sales_location_id": 38244,
"terminal_id": 10353,
"hardware_id": "a0f6fd72e785",
"amount": 1200,
"cashback_amount": 0,
"refundable_amount": 1200,
"currency": 978,
"cardholder_verification": "PIN",
"entry_mode": "chip",
"card_number_customer": "************0035",
"card_number_merchant": "411122******0035",
"card_scheme": "VI",
"card_country": 246,
"card_currency": 978,
"acquirer_timestamp": "161012140619",
"acquirer_reference": "161012123253",
"order_id": "5e98ebdeb3a17b48012400746a18ca9a",
"order_description": "5e98ebdeb3a17b48012400746a18ca9a",
"browser_ip_country": 246,
"authorized": true,
"authorized_amount": 1200,
```

```
    "authorization_code": "string",
    "id_check_performed": true,
  - "external_data": {
        "name": "John Doe",
      + "shift": { … }
    },
    "referring_transactions": "2a:1087143945"
}
```