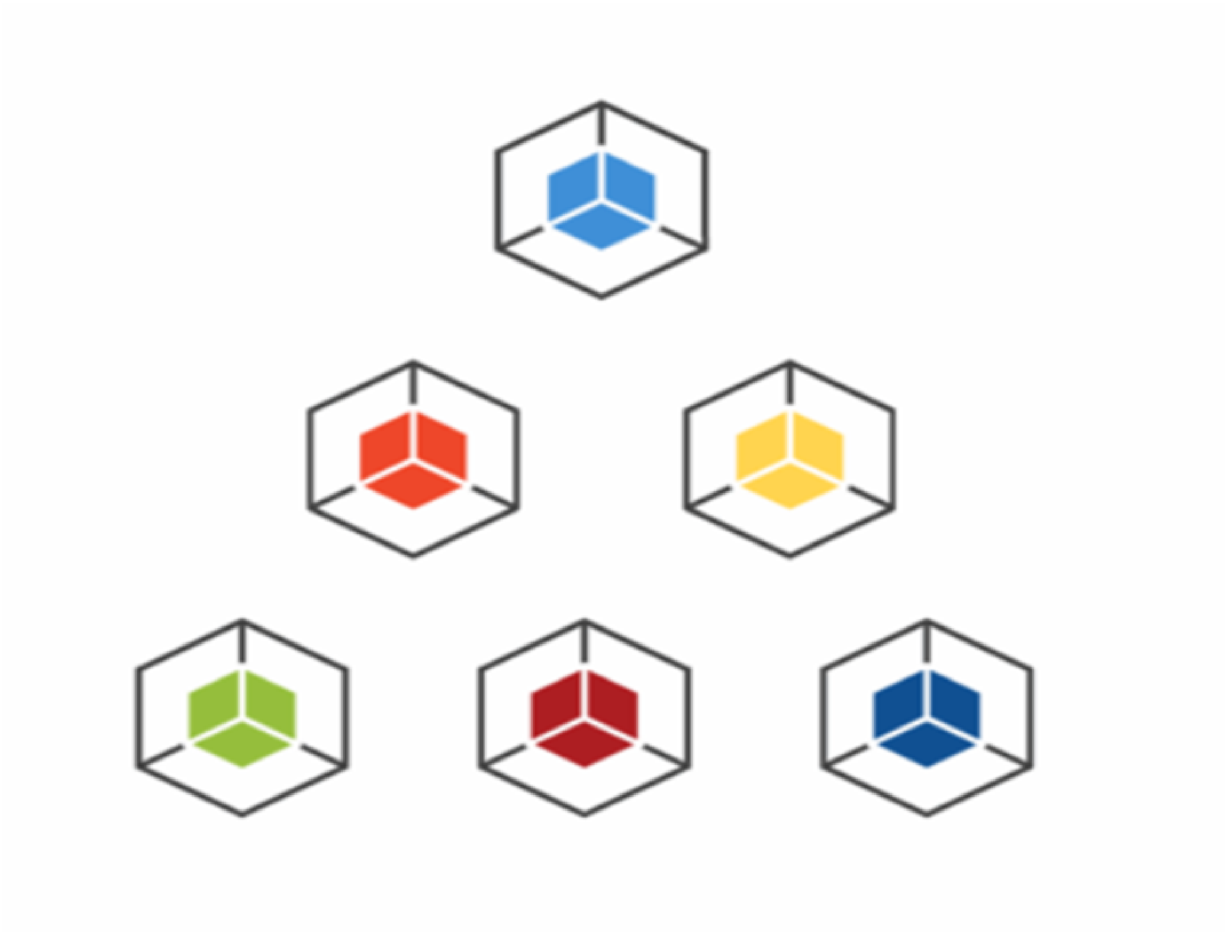
PayGate API V1.0.0

November 13, 2025



Amendments

Version	Date	Description
1.0.0	November 13, 2025	Initial release of the API specification.

Contents

1 Integration: Narratives **4**

2 API Operations **5**

2.1 make-payment 5

2.1.1 Request 5

2.1.2 INPUT Parameters for the request body: 5

2.1.3 Success Response 6

2.1.4 Failed Response 6

2.2 make-refund 7

2.2.1 Request 7

2.2.2 INPUT Parameters for the request body: 7

2.2.3 Success Response 8

2.2.4 Failed Response 8

2.3 HTTP Codes 9

2.4 Application Codes 9

1 Integration: Narratives

The purpose of this document is to provide a high-level specification for the Merchant Onboard integration's API (Application Programming Interface). We expect the readers of this document will already have an understanding of basic API/Web Services concepts.

Once the API is called by the client, we will validate and authorize the client and return necessary information based on the level of access they have and information they require. The response will be in JSON format by default.

There will be only one web service URL exposed and that particular web service will have various methods for clients to decide and call based on the data they require.

The web methods described below are some of the possible scenarios for the API calls and it can be extended to create as many web methods as required to service the customer needs.

2 API Operations

2.1 make-payment

The Payment API is designed to securely process customer payments through the merchant's banking gateway. To initiate a payment, the client sends a POST request containing a JSON payload with the required parameters. The key input parameters include `merchant_id` (the unique identifier of the merchant making the request), `order_id` (a unique reference for the customer order), `amount` (the payment amount to be charged), and card details including `card_no`, `card_holder_name`, `card_cvv`, and `card_exp` (expiry date). Once the request is submitted, the API validates all inputs to ensure correctness and communicates with the bank to process the transaction. The response returned is a structured JSON object containing the transaction status, a unique `transaction_id`, the processed amount, and a descriptive message indicating success or failure. This ensures that both the merchant and customer receive immediate feedback on the payment outcome.

2.1.1 Request

Method	**URL**
POST	`Test:http://127.0.0.1:8000/api/make-payment` `Production:N/A`

2.1.2 INPUT Parameters for the request body:

Request Parameter	**Values**	**Condition**	**Explanation**	**Example**
`merchant_id`	integer	Mandatory	Unique identifier of the merchant initiating the transaction	1
`invoice_id`	string	Mandatory	Unique invoice reference for the transaction	INV-12345
`currency_code`	string	Mandatory	Currency type in which the transaction is processed	USD
`card_no`	string	Mandatory	Customer's card number used for the payment	1234123412341234
`card_holder_name`	string	Mandatory	Name of the card holder as printed on the card	John Doe
`card_cvv`	string	Mandatory	3-digit CVV code from the back of the card	123
`card_exp`	string	Mandatory	Expiry date of the card in MMYY format	1225
`amount`	decimal	Mandatory	Total amount to be charged in the transaction	500

Headers	Content-Type: application/json Accept: application/json
Request	`{ "merchant_id": 1, "invoice_id": "INV-12345", "currency_code": "USD", "card_no": "1234123412341234", "card_holder_name": "John Doe", "card_cvv": "123", "card_exp": "1225", "amount": 500 }`

2.1.3 Success Response

Status	**Response**
200	`{ "response_code": 100, "response_message": "Success", "data": { "bank_order_id": "DBOI1763109463965", "order_id": "DBOI1763109463965", "invoice_id": "INV-12345", "amount": 500, "currency_code": "USD", "payment_at": "2025-11-14 08:37:43", "fee_details": { "commission_percentage": "2.50", "commission_fixed": "0.50", "bank_fee": "0.20" } } }`

2.1.4 Failed Response

Status	**Response**
200	`{ "response_code": 4, "response_message": "Wrong bank credentials", "data": [] }`

2.2 make-refund

The Refund API allows merchants to refund a previously completed transaction either partially or fully. Clients make a POST request with a JSON body containing the necessary input parameters: `merchant_id` (identifying the merchant initiating the refund), `order_id` (the original order reference for which the refund is requested), and `refund_amount` (the amount to be refunded). Upon receiving the request, the API validates the refund amount against the original transaction and then communicates with the bank to process the refund. The response is a structured JSON object containing the refund status, a unique `refund_id`, the refunded amount, and a message confirming whether the refund was successful or failed. This provides merchants with a reliable way to manage financial reversals while ensuring proper transaction tracking and reporting.

2.2.1 Request

Method	**URL**
POST	`Test:http://127.0.0.1:8000/api/make-refund` `Production:N/A`

2.2.2 INPUT Parameters for the request body:

Sample Request Parameter	**Values**	**Condition**	**Explanation**	**Example**
`merchant_id`	integer	Mandatory	Unique identifier of the merchant requesting the refund	5
`order_id`	string	Mandatory	Unique reference of the original order to be refunded	JoCI4OmrK6
`refund_amount`	integer	Mandatory	Amount to be refunded for the given order	1

Headers	Content-Type: application/json Accept: application/json

Request	`{`
	`    "merchant_id": 5,`
	`    "order_id": "JoCI4OmrK6",`
	`    "refund_amount": 1`
	`}`

2.2.3 Success Response

Status	Response
200	`{ "response_code": 100, "response_message": "Success", "data": { "bank_order_id": "JoCI4OmrK6", "amount": 1, "currency_code": "BDT", "payment_at": "2025-11-14 09:09:15", "fee_details": { "commission_percentage": "0.00", "commission_fixed": "0.00", "bank_fee": "0.00" }, "net_amount": 1 } }`

2.2.4 Failed Response

Status	Response
200	`{ "response_code": 6, "response_message": "Invoice already exists", "data": [] }`

2.3 HTTP Codes

These are the generic server status codes.

HTTP Code	Description
200	OK
201	Created
202	Accepted (Request accepted, and queued for execution)
400	Bad request
401	Authentication failure
403	Forbidden
404	Resource not found
405	Method Not Allowed
409	Conflict
412	Precondition Failed
413	Request Entity Too Large
500	Internal Server Error
501	Not Implemented
503	Service Unavailable

2.4 Application Codes

These are the custom status codes of this project.

Status Code	Description
100	Successful Operation
1	Basic Validation
6	Invalid Credential