API: Developer Integration Guide

1. Document Metadata

- **Project:** Bloom Pay Financial Gateway v2.4
- **Status:** Approved / Production-Ready
- **Base URL:** https://api.bloompay.io/v2
- **Auth Mechanism:** Bearer Token (OAuth 2.0 / Mutual TLS)

2. System Architecture & Flow

The Bloom Pay architecture utilizes an asynchronous event-driven model to ensure ledger integrity even during high network latency.

- **Idempotency:** Every write request requires an Idempotency-Key in the header to prevent double-charging in the event of a retry.
- **Double-Entry Ledger:** Every transaction moves value from a source_id to a destination_id; funds are never created or destroyed, only reallocated.

3. Core Resource: The Transaction Object

The Transaction object is the atomic unit of the Bloom Pay system.

Attribute	Type	Description
id	String (Prefix: tx_)	Unique identifier for the transaction.
amount	Integer	Amount in the smallest currency unit (cents, dollars, etc).
currency	String (ISO 4217)	Three-letter currency code (USD, EUR, GBP, etc).
status	Enum	pending, succeeded, failed, reversed.
metadata	JSON Object	Custom key-value pairs for internal tracking.

4. API Endpoints & Implementation

4.1 Initiate a Transfer

POST /v2/transfers

This endpoint initiates a move of funds between two internal accounts or to an external routing number.

Request Header:

HTTP

```
Authorization: Bearer <YOUR_TOKEN>
Idempotency-Key: <UUID_V4>
Content-Type: application/json
```

Request Body:

JSON

```
{
  "source_account": "acc_88231",
  "destination_account": "acc_99420",
  "amount": 5000,
  "currency": "USD",
  "description": "Invoice #1024 Payment",
  "metadata": {
    "customer_id": "cust_550"
  }
}
```

4.2 Webhook Notifications

Because financial settlements can take time (especially for ACH or Cross-border), implement a Webhook listener to handle status updates.

Event Type: transfer.succeeded

JSON

```
{
  "id": "evt_2291",
  "type": "transfer.succeeded",
  "data": {
    "object": "transfer",
    "id": "tx_44102",
    "status": "succeeded"
  }
}
```

5. Security & Compliance

- **PCI-DSS Compliance:** Card data must never touch external servers. Use the Bloom-Vault JS library to tokenize sensitive data before hitting the API.

- **Rate Limiting:** 100 requests per second (RPS) per API Key. Exceeding this limit triggers a 429 Too Many Requests response.
- **Signature Verification:** All incoming webhooks must be verified using the Bloom-Signature header to ensure the payload was changed.

6. Error Handling

Bloom Pay uses standard HTTP status codes combined with internal error codes for debugging.

Code	Type	Meaning
402	insufficient_funds	The source account does not have the required balance.
403	restricted_account	The account is frozen due to KYC (Know Your Customer) requirements.
409	idempotency_error	The Idempotency-Key has been used with different parameters.

7. Sandbox Testing

Developers should use the test_ prefix API keys.

- **Simulated Failures:** Passing an amount of 0.02 will trigger a simulated insufficient_funds error.
- **Simulated KYC:** Using the name Regulatory Test will trigger a manual review status.