ProlificEx Crypto Gateway API - Workflow Document

Generated from WORKFLOW.md. This PDF explains the current Laravel API workflows, testing order, authentication, deposits, balances, withdrawals, webhooks, subscriptions, and production gaps.

Table of Contents

This document explains how the current Laravel API works end to end. It is based on the current codebase and routes.

Base local URL:

```
http://127.0.0.1:8001
```

API prefix:

```
/api/v1
```

Health endpoint:

```
GET /api/health/ready
```

1. Main Idea

This project is a crypto payment gateway for merchants.

It is closer to NOWPayments than to Binance or Trust Wallet.

The normal business flow is:

```
Merchant registers
        |
        v
Merchant verifies email
        |
        v
Admin approves merchant
        |
        v
Merchant creates API key
        |
        v
Merchant creates invoice/payment
        |
        v
Customer sends crypto to invoice address
        |
        v
System detects blockchain payment
        |
        v
Merchant balance increases
        |
        v
Merchant can withdraw by creating payout
        |
        v
Admin/operator approves payout
        |
        v
System broadcasts withdrawal
```

Important difference:

```
Current system = merchant payment gateway
Not yet = customer wallet app with user-to-user transfers
```

2. Main Database Concepts

merchants

A merchant is the account owner using the gateway.

Merchant statuses:

```
pending
active
declined
```

A merchant must be:

```
email verified
active/admin approved
```

before using the main API.

api_keys

Merchants create API keys after login.

The public key is used in:

```
x-api-key
```

The secret is used to generate:

```
x-api-signature
```

The secret is shown once only.

assets

Assets are supported currencies/networks.

Current seeded asset:

```
USDTTRC20
Network: tron
Decimals: 6
```

invoices

Invoices are payment requests.

A payment and an invoice are currently almost the same thing in the code.

Payment IDs look like:

```
pay_01K...
```

wallet_addresses

These are blockchain deposit addresses generated for invoices.

In current code, addresses are mostly invoice-based:

```
one invoice -> one payment/deposit address
```

ledger_accounts and ledger_entries

Balances are not stored as simple columns.

Balances are calculated from ledger entries.

Important ledger account types:

```
merchant_available
merchant_reserved
merchant_pending
blockchain_clearing
platform_fee_revenue
network_fee_expense
```

payouts

Payout means withdrawal.

A payout sends merchant balance to an external crypto wallet.

Payout IDs look like:

```
po_01K...
```

payout_batches

Batch payout creates many payout items at once.

Batch IDs look like:

```
pob_01K...
```

3. Authentication Types

There are two authentication styles.

A. Dashboard/Auth APIs use Bearer Token

Used for:

```
login
me
api key creation
webhook endpoint setup
2FA
operator/admin routes
```

Header:

```
Authorization: Bearer LOGIN_TOKEN
Accept: application/json
```

B. Payment APIs use HMAC API Key

Used for:

```
currencies
estimate
invoices
payments
checkout API
balances
payouts
payout batches
subscriptions
addresses
sub-merchants
webhooks/test
```

Required headers:

```
x-api-key: YOUR_PUBLIC_KEY
x-api-timestamp: UNIX_SECONDS
x-api-nonce: UNIQUE_RANDOM_VALUE
x-api-signature: v1=HMAC_SHA256_SIGNATURE
Accept: application/json
Content-Type: application/json
```

For create/update operations, also use:

```
Idempotency-Key: unique-key-1001
```

The HMAC signature is generated from:

```
METHOD
PATH_WITH_QUERY
TIMESTAMP
NONCE
SHA256(EXACT_RAW_BODY)
```

Example canonical string:

```
POST
/api/v1/invoices
1783946909
01KABC...
<sha256 body hash>
```

4. Merchant Onboarding Workflow

Step 1: Register Merchant

```
POST /api/v1/auth/register
```

Body:

```
{
  "name": "Test Merchant",
  "email": "merchant@example.com",
  "password": "Password123456"
}
```

What happens:

```
merchant is created
status is pending
verification email is sent
```

Step 2: Verify Email

The user clicks email link:

```
GET /api/v1/email/verify/{id}/{hash}?expires=...&signature=...
```

What happens:

```
email_verified_at is set
merchant status can still be pending
```

Email verification does not approve the merchant.

Step 3: Admin Approves Merchant

Admin/operator logs in and calls:

```
POST /api/v1/operator/merchants/{merchantId}/approve
```

What happens:

```
merchant status becomes active
```

Decline endpoint:

```
POST /api/v1/operator/merchants/{merchantId}/decline
```

Step 4: Merchant Login

```
POST /api/v1/auth/login
```

Body:

```
{
  "email": "merchant@example.com",
  "password": "Password123456"
}
```

Response includes:

```
merchant
token
```

This token is used for dashboard-style APIs.

5. API Key Workflow

Create API Key

Requires Bearer login token.

```
POST /api/v1/api-keys
Authorization: Bearer LOGIN_TOKEN
```

Body:

```
{
  "environment": "sandbox",
  "scopes": [
    "payments:read",
    "payments:write",
    "balances:read",
    "payouts:read",
    "payouts:write"
  ]
}
```

Response:

```
{
  "public_key": "pk_test_xxx",
  "secret": "sk_test_xxx"
}
```

Use:

```
public_key -> x-api-key
secret -> HMAC signing secret
```

List API Keys

```
GET /api/v1/api-keys
Authorization: Bearer LOGIN_TOKEN
```

Rotate API Key

```
POST /api/v1/api-keys/{apiKey}/rotate
Authorization: Bearer LOGIN_TOKEN
```

Revoke API Key

```
DELETE /api/v1/api-keys/{apiKey}
Authorization: Bearer LOGIN_TOKEN
```

6. Currencies Workflow

Get Supported Currencies

Requires HMAC.

```
GET /api/v1/currencies
```

Current expected response includes:

```
{
  "code": "USDTTRC20",
  "name": "Tether USD",
  "network": "tron",
  "decimals": 6,
  "minimum_payment": "1.000000",
  "enabled": true
}
```

Source code:

```
CurrencyController
Asset model
assets table
```

7. Estimate Workflow

Estimate tells merchant how much crypto is needed for a USD amount.

```
POST /api/v1/estimate
```

Body:

```
{
  "amount": "10",
  "currency_from": "USD",
  "currency_to": "USDTTRC20"
}
```

Current quote logic:

```
QuoteService
USDT/USD fixed rate from config
```

Current project is MVP. It does not yet use a live exchange-rate provider.

8. Invoice / Deposit Workflow

Deposit means customer pays a merchant invoice.

Full Deposit Flow

```
Merchant creates invoice
        |
        v
InvoiceService creates invoice row
        |
        v
BlockchainGateway creates deposit address
        |
        v
wallet_addresses row is created
        |
        v
Customer sends crypto to that address
        |
        v
Polling/webhook detects transaction
        |
        v
Trc20TransactionProcessor links transaction to invoice
        |
        v
PaymentFinalizer posts ledger entries
        |
        v
Merchant available balance increases
        |
        v
Webhook payment.finished is queued
```

Create Invoice

Requires HMAC.

```
POST /api/v1/invoices
Idempotency-Key: inv-1001
```

Body:

```
{
  "price_amount": "10.00",
  "price_currency": "USD",
  "pay_currency": "USDTTRC20",
  "order_id": "ORDER-1001",
  "payer": {
    "name": "Test Customer",
    "email": "customer@example.com"
  },
  "items": [
    {
      "name": "Test item",
      "quantity": 1,
      "unit_amount": "10.00"
    }
  ]
}
```

Important response fields:

```
invoice_id/payment_id
pay_amount
pay_currency
pay_address
checkout_url
status
expires_at
```

List Invoices

```
GET /api/v1/invoices
```

Optional filters:

```
status
order_id
per_page
```

Get Invoice

```
GET /api/v1/invoices/{invoiceId}
```

Send Invoice Email

```
POST /api/v1/invoices/{invoiceId}/send-email
```

Body optional:

```
{
  "email": "customer@example.com"
}
```

Uses configured mailer. In your project, Resend is configured.

Download Invoice PDF

```
GET /api/v1/invoices/{invoiceId}/pdf
```

This returns a simple generated PDF response.

9. Checkout Workflow

Checkout is the customer-facing payment page or API.

Browser Checkout Page

Open in browser:

```
http://127.0.0.1:8001/checkout/{paymentId}
```

Example:

```
http://127.0.0.1:8001/checkout/pay_01KXXXXX
```

This shows:

```
order
USD amount
crypto amount
network
payment address
status
expiry
```

Checkout JSON API

Requires HMAC.

```
GET /api/v1/checkout/{paymentId}
```

Select Currency

```
POST /api/v1/checkout/{paymentId}/select-currency
```

Body:

```
{
  "pay_currency": "USDTTRC20"
}
```

Current sandbox only supports the original invoice currency.

10. How Blockchain Deposit Detection Works

Current detection classes:

```
DispatchOpenInvoicePollingJob
PollOpenInvoiceAddressJob
Trc20TransactionProcessor
RefreshTransactionFinalityJob
FinalizePaymentJob
PaymentFinalizer
```

Polling Flow

```
DispatchOpenInvoicePollingJob
        |
        v
Find waiting/confirming invoices
        |
        v
PollOpenInvoiceAddressJob
        |
        v
BlockchainGateway::listAddressTransactions
        |
        v
Trc20TransactionProcessor::process
```

Transaction Processor

`Trc20TransactionProcessor` checks:

```
network matches asset network
contract address matches token contract
token decimals match asset decimals
amount is positive
destination address exists in wallet_addresses
invoice matches asset/environment
```

Then it creates/updates:

```
chain_transactions
invoice_transactions
invoice.paid_amount_atomic
```

Then invoice status changes:

```
waiting -> confirming
```

Finalizer

`PaymentFinalizer` checks confirmations.

Then posts ledger entries.

Deposit ledger movement:

```
blockchain_clearing debit
merchant_available credit
platform_fee_revenue credit
```

Simple meaning:

```
Merchant receives crypto balance minus platform fee.
```

11. Balance Workflow

Get Balance

Requires HMAC.

```
GET /api/v1/balances
```

Source code:

```
BalanceController
LedgerService::merchantBalances()
```

Balances are calculated from ledger entries.

Important balance types:

```
merchant_available = can withdraw
merchant_reserved = locked for payout
merchant_pending = future/pending balance, not heavily used yet
```

Example:

```
Customer paid 10 USDT
Platform fee 0.05 USDT
Merchant available becomes 9.95 USDT
```

12. Withdrawal / Payout Workflow

Withdraw is called payout in the code.

Full Withdraw Flow

```
Merchant has available balance
        |
        v
Merchant creates payout
        |
        v
System checks 2FA
        |
        v
System checks address/currency/amount
        |
        v
System checks merchant_available balance
        |
        v
Ledger moves available -> reserved
        |
        v
Payout enters screening
        |
        v
Payout enters approval
        |
        v
Operator approves payout
        |
        v
BroadcastPayoutJob sends transfer to provider
        |
        v
SyncPayoutJob checks provider status
        |
        v
PayoutSettlementService completes payout
```

Create Payout

Requires HMAC.

```
POST /api/v1/payouts
Idempotency-Key: payout-1001
```

Body:

```
{
  "pay_currency": "USDTTRC20",
  "to_address": "TARGET_EXTERNAL_WALLET_ADDRESS",
  "amount": "5.00",
  "two_factor_code": "123456"
}
```

Checks in `PayoutController`:

```
merchant two_factor_enabled must be true
currency must exist and be enabled
to_address must pass provider validation
amount must be above minimum
merchant_available must have enough balance
```

Ledger Movement When Payout Is Created

Before:

```
merchant_available = 10 USDT
merchant_reserved = 0 USDT
```

Merchant withdraws 5 USDT.

Ledger posts:

```
merchant_available debit 5
merchant_reserved credit 5
```

After:

```
merchant_available = 5 USDT
merchant_reserved = 5 USDT
```

Meaning:

```
Money is locked so merchant cannot withdraw it twice.
```

Approve Payout

Requires admin/operator Bearer token.

```
POST /api/v1/operator/payouts/{payoutId}/approve
```

Broadcast Payout

After approval:

```
ApprovePayoutJob
        |
        v
BroadcastPayoutJob
        |
        v
BlockchainGateway::createTransfer
```

For real TRON provider:

```
SelfHostedTronGateway calls signer service /v1/transfers
```

For local sandbox:

```
DisabledBlockchainGateway returns fake pending transfer
```

Complete Payout

`SyncPayoutJob` checks provider transfer status.

When completed, `PayoutSettlementService::complete()` posts:

```
merchant_reserved debit
blockchain_clearing credit
network_fee_expense debit
```

```
blockchain_clearing credit
```

Simple meaning:

```
Reserved money is finally sent out.
Network fee is recorded.
```

13. Payout Statuses

Possible statuses used by current code:

```
pending_screening
manual_review
pending_approval
approved
broadcasting
confirming
completed
failed
```

Common happy path:

```
pending_screening -> pending_approval -> approved -> broadcasting -> confirming -> completed
```

Failure/manual path:

```
pending_screening -> manual_review
broadcasting -> manual_review
confirming -> failed
```

14. Batch Payout Workflow

Batch payout means multiple withdrawals in one request.

Create Batch

Requires HMAC.

```
POST /api/v1/payout-batches
```

Body:

```
{
  "pay_currency": "USDTTRC20",
  "items": [
    {
      "to_address": "WALLET_1",
      "amount": "1.50"
    },
    {
      "to_address": "WALLET_2",
      "amount": "2.00"
    }
  ]
}
```

What happens:

```
payout_batches row is created
payout_batch_items rows are created
batch status = pending_approval
```

List Batches

```
GET /api/v1/payout-batches
```

Get Batch

```
GET /api/v1/payout-batches/{batchId}
```

Approve Batch

```
POST /api/v1/payout-batches/{batchId}/approve
```

Current behavior:

```
creates individual payout records
marks batch approved
items become created
```

Important note:

The batch approval endpoint currently creates payout records in `pending_approval`. It does not fully broadcast them by itself. The individual operator payout approval flow still handles actual payout approval/broadcasting.

15. Webhook Workflow

Merchants can configure webhook endpoints.

Create Webhook Endpoint

Requires Bearer login token.

```
POST /api/v1/webhook-endpoints
Authorization: Bearer LOGIN_TOKEN
```

Body:

```
{
  "url": "https://example.com/webhook",
  "events": ["payment.finished"],
  "environment": "sandbox"
}
```

Response includes webhook signing secret:

```
whsec_xxx
```

Test Webhook

Requires HMAC.

```
POST /api/v1/webhooks/test
```

What happens:

```
creates webhook delivery
queues DeliverMerchantWebhookJob
```

Replay Webhook

Requires Bearer login token.

```
POST /api/v1/webhook-deliveries/{deliveryId}/replay
```

Payment Finished Webhook

When payment finalizes, `PaymentFinalizer` creates payload:

```
{
  "type": "payment.finished",
  "data": {
    "payment_id": "pay_xxx",
    "order_id": "ORDER-1001",
    "status": "finished",
    "pay_currency": "USDTTRC20",
    "pay_amount": "10.000000",
    "actually_paid": "10.000000",
    "tx_hashes": []
  }
}
```

16. Subscription Workflow

Subscriptions are an MVP layer currently.

They create plan/subscription records. They do not yet automatically generate recurring invoices by scheduler.

Create Plan

Requires HMAC.

```
POST /api/v1/subscriptions/plans
```

Body:

```
{
  "name": "Monthly Plan",
  "price_amount": "12.00",
  "price_currency": "USD",
  "pay_currency": "USDTTRC20",
  "interval": "month"
}
```

List Plans

```
GET /api/v1/subscriptions/plans
```

Create Subscription

```
POST /api/v1/subscriptions
```

Body:

```
{
  "plan_id": "plan_xxx",
  "customer": {
    "name": "Subscriber",
    "email": "subscriber@example.com"
  }
}
```

List Subscriptions

```
GET /api/v1/subscriptions
```

Cancel Subscription

```
POST /api/v1/subscriptions/{subscriptionId}/cancel
```

17. Two-Factor Authentication Workflow

2FA is important because payouts require it.

Setup 2FA

Requires Bearer token.

```
POST /api/v1/2fa/setup
```

Returns secret/QR data.

Enable 2FA

```
POST /api/v1/2fa/enable
```

Body:

```
{
  "code": "123456"
}
```

Step-Up 2FA

```
POST /api/v1/2fa/step-up
```

Body:

```
{
  "code": "123456"
}
```

Payouts require:

```
merchant.two_factor_enabled = true
```

18. Operator/Admin Workflow

Admin/operator routes require:

```
Bearer login token
merchant active
merchant verified
operator email configured in GATEWAY_OPERATOR_EMAILS
```

Operator Dashboard

```
GET /api/v1/operator/dashboard
```

Treasury Dashboard

```
GET /api/v1/operator/treasury/dashboard
```

Merchant Approval

```
GET  /api/v1/operator/merchants
POST /api/v1/operator/merchants/{merchantId}/approve
POST /api/v1/operator/merchants/{merchantId}/decline
```

Payout Approval

```
POST /api/v1/operator/payouts/{payoutId}/approve
```

19. Address Admin Workflow

List Addresses

Requires HMAC.

```
GET /api/v1/addresses
```

This lists `wallet_addresses` belonging to merchant/environment.

Update Address

```
PATCH /api/v1/addresses/{id}
```

Body:

```
{
  "alias": "Main invoice address",
  "status": "active"
}
```

Current address system is mostly invoice deposit addresses.

20. Current Local/Sandbox Behavior

Because real blockchain provider may not be configured, local sandbox has fallback behavior.

Deposit Address

If provider fails in sandbox, invoice creation creates fake address:

```
TST + hash
```

This lets Postman tests work without real TRON signer service.

Payout Transfer

If using disabled/sandbox gateway, transfers are fake/pending.

For real withdrawals, you need:

```
BLOCKCHAIN_DRIVER=self_hosted_tron
TRON_SIGNER_URL=...
TRON_SIGNER_TOKEN=...
TRON_NODE_URL=...
```

and a real signer/provider service implementing:

```
/v1/addresses
/v1/transfers
/wallet/validateaddress
```

21. What Is Working Now

These flows were smoke tested:

```
invoice create -> 201
checkout API -> 200
subscription plan create -> 201
subscription create -> 201
payout batch create -> 201
payout batch approve -> 200
```

Migrations also complete:

```
php artisan migrate
INFO Nothing to migrate.
```

22. What Is Not Fully Built Yet

Real blockchain provider integration

The interfaces exist, but production requires a real provider/signer.

Missing/needs production hardening:

```
real TRON address generation
real TRON transfer broadcasting
real transaction polling
real webhook verification
hot wallet management
private key custody/security
```

Customer wallet app

Current system is merchant gateway, not user wallet app.

Missing APIs for wallet app:

```
GET  /api/v1/wallets
POST /api/v1/wallets/deposit-address
GET  /api/v1/wallets/transactions
POST /api/v1/wallets/transfer-internal
POST /api/v1/wallets/withdraw
```

Internal transfers

No current user-to-user transfer flow.

Automatic recurring subscription billing

Plans/subscriptions exist, but there is no scheduler creating invoices every billing period yet.

Full dashboard frontend

Backend routes exist, but no full merchant/admin dashboard UI yet.

23. Postman Testing Order

Recommended testing order:

```
1. Register merchant
2. Verify email
3. Admin approve merchant
4. Login merchant
5. Create API key
6. Configure HMAC script in Postman
7. GET currencies
8. POST estimate
9. POST invoices
10. Open checkout page
11. GET balances
12. Setup/enable 2FA
13. POST payouts
14. Admin approve payout
15. Create webhook endpoint
16. Test webhook
17. Create subscription plan
18. Create subscription
19. Create payout batch
20. Approve payout batch
```

24. Minimal HMAC Postman Script

Set environment variables:

```
api_key = pk_test_xxx
api_secret = sk_test_xxx
```

Pre-request script:

```
const apiKey = pm.environment.get("api_key");
const apiSecret = pm.environment.get("api_secret");

if (!apiKey || !apiSecret) {
  throw new Error("Set api_key and api_secret first.");
}

const timestamp = Math.floor(Date.now() / 1000).toString();
const nonce = crypto.randomUUID();
const method = pm.request.method.toUpperCase();
const path = pm.request.url.getPathWithQuery();
const body = pm.request.body && pm.request.body.raw ? pm.request.body.raw : "";
const bodyHash = CryptoJS.SHA256(body).toString();
const canonical = [method, path, timestamp, nonce, bodyHash].join("\n");
const signature = CryptoJS.HmacSHA256(canonical, apiSecret).toString();

pm.request.headers.upsert({ key: "x-api-key", value: apiKey });
pm.request.headers.upsert({ key: "x-api-timestamp", value: timestamp });
pm.request.headers.upsert({ key: "x-api-nonce", value: nonce });
pm.request.headers.upsert({ key: "x-api-signature", value: `v1=${signature}` });
pm.request.headers.upsert({ key: "Accept", value: "application/json" });
pm.request.headers.upsert({ key: "Content-Type", value: "application/json" });
```

25. Short Mental Model

Use this simple model:

```
Merchant = business account
API key = merchant integration credential
Invoice = request customer to pay
Checkout = page/API showing payment instructions
WalletAddress = crypto address for invoice deposit
ChainTransaction = detected blockchain transaction
Ledger = internal balance accounting
Balance = sum of ledger entries
Payout = merchant withdrawal
Webhook = notification to merchant system
Operator = admin who approves merchants/payouts
```

Deposit:

```
Customer pays invoice -> merchant_available increases
```

Withdraw:

```
Merchant creates payout -> available moves to reserved -> admin approves -> crypto sent out
```

That is the current project workflow.