Wallet & Transactions

Author: Josh S. Sakweli, Backend Lead Team
Last Updated: 2025-10-02
Version: v1.0

Base URL:

Short Description: The Financial System manages all monetary transactions using double-entry bookkeeping. It handles user wallets, escrow for secure payments, transaction tracking, and payment processing with complete auditability.

Core Components

1. Ledger System

Double-entry bookkeeping where every transaction has equal debits and credits. Maintains user wallets, escrow accounts, platform revenue, and external flow tracking.

2. Escrow Management

Holds buyer payments until delivery confirmation. Protects both parties and automatically splits funds: 95% to seller, 5% platform fee.

3. Transaction History

User-facing records of all financial activity including purchases, refunds, and wallet operations.

Payment Flow

1. Validate checkout and payment method
2. Move funds to escrow via ledger
3. Create transaction records
4. Generate order on success
5. Release funds after delivery (seller 95%, platform 5%)

Payment Methods

Active: WALLET, CASH_ON_DELIVERY

Planned: Mobile money (M-Pesa, Tigo Pesa, Airtel Money), cards, bank transfers

Technical Specifications

- Currency: TZS (Tanzanian Shillings)
- Precision: BigDecimal (15 digits, 2 decimals)
- All operations are atomic transactions
- Complete audit trail maintained

- Wallet
- Transaction History

Wallet

Author: Josh S. Sakweli, Backend Lead Team
Last Updated: 2026-05-21
Version: v1.1

Base URL: `https://apinexgate.glueauth.com/api/v1/wallet`

Short Description: The Wallet API manages user digital wallets for storing and using funds within the platform. It handles wallet creation, balance queries, checkout balance readiness checks, top-ups from external sources, withdrawals to external accounts, and wallet status management. All balances are maintained through the underlying ledger system ensuring accurate double-entry bookkeeping.

Hints:

- Wallets are automatically created for users on first access
- All monetary values are in TZS (Tanzanian Shillings)
- Actual balance is stored in the ledger system, not the wallet entity
- Wallets can be deactivated for security reasons
- PSP (Selcom) minimum top-up amount is **1,000 TZS** — you cannot deposit less than this via mobile money or card
- All operations create transaction history entries
- Top-ups represent money entering the platform from external sources (M-Pesa, Selcom, bank transfer, etc.)
- Withdrawals represent money leaving the platform to external accounts
- Each user can have only one wallet
- The `checkout-balance-check` endpoint is the recommended way to determine if a user needs to top up before paying — it accounts for the PSP minimum automatically

---

Standard Response Format

All API responses follow a consistent structure using our Globe Response Builder pattern:

Success Response Structure

```
{
  "success": true,
  "httpStatus": "OK",
```

```
  "message": "Operation completed successfully",
  "action_time": "2025-10-02T10:30:45",
  "data": { }
}
```

Error Response Structure

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Error description",
  "action_time": "2025-10-02T10:30:45",
  "data": "Error description"
}
```

Standard Response Fields

Field	Type	Description
`success`	boolean	Always `true` for successful operations, `false` for errors
`httpStatus`	string	HTTP status name (OK, CREATED, BAD_REQUEST, NOT_FOUND, etc.)
`message`	string	Human-readable message describing the operation result
`action_time`	string	ISO 8601 timestamp of when the response was generated
`data`	object/string	Response payload for success, error details for failures

---

HTTP Method Badge Standards

For better visual clarity, all endpoints use colored badges for HTTP methods with the following standard colors:

- **GET** - GET - Green (Safe, read-only operations)
- **POST** - POST - Blue (Create new resources)
- **PUT** - PUT - Yellow (Full updates)

Endpoints

1. Get My Wallet

Purpose: Retrieves the authenticated user's wallet information including wallet ID, account details, current balance, and status. If wallet doesn't exist, it is automatically created.

Endpoint: `GET` `{base_url}/my-wallet`

Access Level: □□ Protected (Requires Authentication)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated user

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Wallet retrieved successfully",
  "action_time": "2025-10-02T10:30:45",
  "data": {
    "walletId": "w1a2l3l4-e5t6-7890-abcd-ef1234567890",
    "accountId": "a1c2c3o4-u5n6-7890-abcd-ef1234567890",
    "accountUserName": "john_doe",
    "currentBalance": 150000.00,
    "isActive": true,
    "createdAt": "2025-09-15T08:20:30",
    "updatedAt": "2025-10-02T10:15:20"
  }
}
```

Success Response Fields:

Field	Description

walletId	Unique identifier for the wallet
accountId	User account UUID this wallet belongs to
accountUserName	Username of the wallet owner
currentBalance	Current wallet balance in TZS (from ledger system)
isActive	Whether the wallet is currently active
createdAt	ISO 8601 timestamp when wallet was created
updatedAt	ISO 8601 timestamp of last wallet update

Error Response Examples:

Unauthorized (401):

```
{
  "success": false,
  "httpStatus": "UNAUTHORIZED",
  "message": "Authentication token is required",
  "action_time": "2025-10-02T10:30:45",
  "data": "Authentication token is required"
}
```

---

2. Get My Balance

Purpose: Retrieves only the current balance of the authenticated user's wallet. This is a lightweight endpoint for quick balance checks.

Endpoint: `GET` `{base_url}/balance`

Access Level: □□ Protected (Requires Authentication)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated user

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Balance retrieved successfully",
  "action_time": "2025-10-02T10:35:45",
  "data": {
    "balance": 150000.00,
    "currency": "TZS"
  }
}
```


Success Response Fields:

Field	Description
balance	Current wallet balance
currency	Currency code (always TZS)

Error Response Examples:

Unauthorized (401):


```
{
  "success": false,
  "httpStatus": "UNAUTHORIZED",
  "message": "Authentication token is required",
  "action_time": "2025-10-02T10:35:45",
  "data": "Authentication token is required"
}
```


---

3. Get Wallet by ID (Admin)

Purpose: Retrieves detailed information about any wallet by its ID. This is an administrative endpoint requiring SUPER_ADMIN or STAFF_ADMIN role, or wallet ownership.

Endpoint: `GET` `{base_url}/{walletId}`

Access Level: □□ Protected (Requires Authentication + Admin Role or Ownership)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated admin or owner

Path Parameters:

Parameter	Type	Required	Description	Validation
walletId	string (UUID)	Yes	Unique identifier of the wallet	Valid UUID format

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Wallet retrieved successfully",
  "action_time": "2025-10-02T10:50:45",
  "data": {
    "walletId": "w1a2l3l4-e5t6-7890-abcd-ef1234567890",
    "accountId": "a1c2c3o4-u5n6-7890-abcd-ef1234567890",
    "accountUserName": "john_doe",
    "currentBalance": 150000.00,
    "isActive": true,
    "createdAt": "2025-09-15T08:20:30",
    "updatedAt": "2025-10-02T10:15:20"
  }
}
```

Error Response Examples:

Not Found - No Permission (404):

```
{
  "success": false,
  "httpStatus": "NOT_FOUND",
  "message": "You do not have permission to access this wallet",
  "action_time": "2025-10-02T10:50:45",
  "data": "You do not have permission to access this wallet"
}
```

4. Activate Wallet (Admin)

Purpose: Activates a previously deactivated wallet, allowing the user to perform transactions again. Requires SUPER_ADMIN role or wallet ownership.

Endpoint: **PUT** `{base_url}/{walletId}/activate`

Access Level: □□ Protected (Requires Authentication + SUPER_ADMIN Role or Ownership)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated admin or owner

Path Parameters:

Parameter	Type	Required	Description	Validation
walletId	string (UUID)	Yes	Unique identifier of the wallet to activate	Valid UUID format

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Wallet activated successfully",
  "action_time": "2025-10-02T10:55:45",
  "data": null
}
```

Error Response Examples:

Not Found - No Permission (404):

```
{
  "success": false,
  "httpStatus": "NOT_FOUND",
```

```
  "message": "You do not have permission to activate this wallet",
  "action_time": "2025-10-02T10:55:45",
  "data": "You do not have permission to activate this wallet"
}
```

---

5. Deactivate Wallet (Admin)

Purpose: Deactivates a wallet for security or administrative reasons. Once deactivated, the wallet cannot perform any transactions until reactivated. Requires SUPER_ADMIN or STAFF_ADMIN role, or wallet ownership.

Endpoint: **PUT** `{base_url}/{walletId}/deactivate`

Access Level: □□ Protected (Requires Authentication + Admin Role or Ownership)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated admin or owner

Path Parameters:

Parameter	Type	Required	Description	Validation
walletId	string (UUID)	Yes	Unique identifier of the wallet to deactivate	Valid UUID format

Query Parameters:

Parameter	Type	Required	Description	Validation
reason	string	Yes	Reason for deactivation	Required, cannot be empty

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
```

```
  "message": "Wallet deactivated successfully",
  "action_time": "2025-10-02T11:00:45",
  "data": null
}
```

Error Response Examples:

Not Found - No Permission (404):

```
{
  "success": false,
  "httpStatus": "NOT_FOUND",
  "message": "You do not have permission to deactivate this wallet",
  "action_time": "2025-10-02T11:00:45",
  "data": "You do not have permission to deactivate this wallet"
}
```

---

6. Checkout Balance Check

Purpose: Checks whether the authenticated user's wallet has enough balance to pay for a specific checkout session (product or event ticket). If balance is insufficient, it returns the exact shortfall and a **PSP-safe recommended top-up amount** — meaning the amount is already rounded up to meet the PSP (Selcom) minimum deposit of 1,000 TZS. The frontend should call this endpoint immediately after a checkout session is created and before initiating wallet payment, so it can show a smart "Top Up to Continue" shortcut instead of sending the user to the wallet screen manually.

Endpoint: `GET` `{base_url}/checkout-balance-check`

Access Level: □□ Protected (Requires Authentication)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated user

Query Parameters:

Parameter	Type	Required	Description	Allowed Values

sessionId	string (UUID)	Yes	The checkout session ID to check against	Valid UUID
domain	string (enum)	Yes	The type of checkout session	`PRODUCT`, `EVENT`

How `recommendedTopUp` is calculated:

```
shortfall        = sessionTotal - walletBalance   (minimum 0, never negative)
recommendedTopUp = max(shortfall, 1000)           (PSP minimum floor)
```

The PSP minimum floor ensures the frontend never suggests an amount the user can't actually deposit through Selcom. If the user is only short 10 TZS, we still recommend 1,000 TZS because Selcom won't process anything below that — the extra will just remain in the wallet for future use.

When `recommendedTopUp` is omitted: When `hasSufficientBalance` is `true`, there is nothing to top up, so `recommendedTopUp` is not included in the response (`@JsonInclude(NON_NULL)`).

Scenario A — Balance is sufficient

> User has **600 TZS**, ticket costs **500 TZS**

Request:

```
GET {base_url}/checkout-balance-check?sessionId=abc123&domain=EVENT
Authorization: Bearer {token}
```

Response:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Checkout balance check completed",
  "action_time": "2026-05-21T09:00:00",
  "data": {
    "walletBalance": 600.00,
    "sessionTotal": 500.00,
    "shortfall": 0.00,
    "hasSufficientBalance": true,
    "pspMinimum": 1000.00,
```

```
    "currency": "TZS"
  }
}
```

Frontend behaviour: `hasSufficientBalance` is `true` → hide top-up prompt, proceed directly to wallet payment.

Scenario B — Shortfall is below PSP minimum (PSP floor kicks in)

> User has **290 TZS**, product costs **300 TZS** → shortfall is only 10 TZS, but PSP minimum is 1,000 TZS

Request:

```
GET {base_url}/checkout-balance-check?sessionId=xyz789&domain=PRODUCT
Authorization: Bearer {token}
```

Response:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Checkout balance check completed",
  "action_time": "2026-05-21T09:05:00",
  "data": {
    "walletBalance": 290.00,
    "sessionTotal": 300.00,
    "shortfall": 10.00,
    "hasSufficientBalance": false,
    "recommendedTopUp": 1000.00,
    "pspMinimum": 1000.00,
    "currency": "TZS"
  }
}
```

Frontend behaviour: Show "Top Up to Continue" prompt pre-filled with **1,000 TZS**. After top-up, wallet will have 1,290 TZS — more than enough. The 990 extra TZS stays in the wallet for future

use.

Scenario C — Shortfall is above PSP minimum (exact shortfall recommended)

> User has **300 TZS**, product costs **2,000 TZS** → shortfall is 1,700 TZS, which is already above the PSP minimum

Request:

```
GET {base_url}/checkout-balance-check?sessionId=def456&domain=PRODUCT
Authorization: Bearer {token}
```

Response:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Checkout balance check completed",
  "action_time": "2026-05-21T09:10:00",
  "data": {
    "walletBalance": 300.00,
    "sessionTotal": 2000.00,
    "shortfall": 1700.00,
    "hasSufficientBalance": false,
    "recommendedTopUp": 1700.00,
    "pspMinimum": 1000.00,
    "currency": "TZS"
  }
}
```


Frontend behaviour: Show "Top Up to Continue" prompt pre-filled with **1,700 TZS**. After top-up, wallet will have exactly 2,000 TZS — just enough to pay.

Scenario D — Session not found

>

> Invalid or expired `sessionId`

Response (404):

```
{
  "success": false,
  "httpStatus": "NOT_FOUND",
  "message": "Product checkout session not found",
  "action_time": "2026-05-21T09:15:00",
  "data": "Product checkout session not found"
}
```

Frontend behaviour: The session may have expired. Redirect the user back to start a new checkout.

Scenario E — Wrong domain for the session ID

> Passing `domain=PRODUCT` for an event session ID (or vice versa)

Response (404):

```
{
  "success": false,
  "httpStatus": "NOT_FOUND",
  "message": "Product checkout session not found",
  "action_time": "2026-05-21T09:20:00",
  "data": "Product checkout session not found"
}
```

Frontend behaviour: Ensure `domain` matches the checkout flow — `EVENT` for ticket purchases, `PRODUCT` for e-commerce.

Success Response Fields:

Field	Type	Present when	Description
walletBalance	number	Always	Authenticated user's current wallet balance in TZS

Field	Type	Present when	Description
sessionTotal	number	Always	Total amount due for the checkout session
shortfall	number	Always	Amount the user is short (`sessionTotal - walletBalance`, minimum 0)
hasSufficientBalance	boolean	Always	`true` if wallet can cover the session total, `false` if not
recommendedTopUp	number	Only when `hasSufficientBalance` is `false`	Amount to suggest to the user — already PSP-floor adjusted (`max(shortfall, 1000)`)
pspMinimum	number	Always	The current PSP minimum deposit amount (1,000 TZS)
currency	string	Always	Always `TZS`

Integration Examples

Example 1: Smart Checkout Payment Flow (Recommended)

This is the correct way to handle payments from v1.1 onwards. Replace the old "check balance manually then guess the top-up amount" approach with this.

Step 1: User creates checkout session (via `/api/v1/checkout` or `/api/v1/event-checkout`)

Step 2: Immediately call balance check

```
GET /api/v1/wallet/checkout-balance-check?sessionId=abc-123&domain=EVENT
Authorization: Bearer {token}
```

Step 3a: Balance sufficient → pay directly

Response has `hasSufficientBalance: true`. Frontend proceeds to wallet payment — no top-up prompt needed.

Step 3b: Balance insufficient → show top-up shortcut

Response has `hasSufficientBalance: false`. Frontend shows an inline modal:

> "Your wallet has 290 TZS. You need 300 TZS for this purchase. Top up 1,000 TZS to continue?" **[Top Up Now] [Cancel]**

The "Top Up Now" button calls the collection initiation endpoint with the pre-filled `recommendedTopUp` value. No redirect to the wallet screen. No guessing.

Step 4: After top-up webhook fires → wallet is credited → user confirms payment → wallet payment proceeds.

Example 2: Top-up and Purchase Flow (Legacy — still works but not recommended)

Step 1: Check balance

```
GET /api/v1/wallet/balance
Authorization: Bearer {token}
```

Response shows: 50,000 TZS (user manually figures out it's not enough for 100,000 TZS purchase)

Step 2: Top-up wallet (user has to guess the amount themselves)

```
POST /api/v1/wallet/topup
Authorization: Bearer {token}
Content-Type: application/json

{
  "amount": 60000.00,
  "description": "M-Pesa top-up"
}
```

Step 3: Confirm new balance

```
GET /api/v1/wallet/balance
Authorization: Bearer {token}
```

Response shows: 110,000 TZS (now sufficient)

>

> ⚠ Prefer **Example 1** for new implementations. Example 2 requires the user to calculate the top-up amount themselves and navigate away to the wallet screen, which is a poor UX.

Example 3: Withdrawal Flow

Step 1: Get wallet info

```
GET /api/v1/wallet/my-wallet
Authorization: Bearer {token}
```

Step 2: Request withdrawal

```
POST /api/v1/wallet/withdraw
Authorization: Bearer {token}
Content-Type: application/json

{
  "amount": 50000.00,
  "description": "Withdraw to CRDB Bank - Account 1234567890"
}
```

Step 3: Verify new balance

```
GET /api/v1/wallet/balance
Authorization: Bearer {token}
```

Rate Limiting

Rate Limits:

- Get Wallet/Balance: 60 requests per minute per user
- Checkout Balance Check: 60 requests per minute per user
- Top-up: 10 requests per minute per user
- Withdraw: 10 requests per minute per user
- Admin Operations: 30 requests per minute per admin

Rate Limit Headers:

```
X-RateLimit-Limit: 10
X-RateLimit-Remaining: 7
X-RateLimit-Reset: 1696258800
```

Rate Limit Exceeded:

```
{
  "success": false,
  "httpStatus": "TOO_MANY_REQUESTS",
  "message": "Rate limit exceeded. Please try again later.",
  "action_time": "2025-10-02T11:20:45",
  "data": "Rate limit exceeded. Please try again later."
}
```

Transaction History

Author: Josh S. Sakweli, Backend Lead Team
Last Updated: 2025-10-02
Version: v1.0

Base URL: `https://apinexgate.glueauth.com/api/v1/transaction-history`

Short Description: The Transaction History API provides user-facing records of all financial activities within the platform. It tracks purchases, refunds, wallet operations, sales earnings, and platform fees with complete details for each transaction. All transactions are linked to underlying ledger entries for full auditability.

Hints:

- All monetary values are in TZS (Tanzanian Shillings)
- Each transaction has a unique reference number (format: #YYYYTNNNNNN)
- Transactions are read-only - they cannot be modified or deleted
- DEBIT transactions represent money leaving the user's wallet (negative)
- CREDIT transactions represent money entering the user's wallet (positive)
- All transactions are linked to ledger entries for audit purposes
- Transactions are automatically created by the system during financial operations
- Pagination is supported with default 20 items per page
- Transactions can be filtered by type, direction, and date range

---

Standard Response Format

All API responses follow a consistent structure using our Globe Response Builder pattern:

Success Response Structure

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Operation completed successfully",
  "action_time": "2025-10-02T10:30:45",
  "data": {
    // Actual response data goes here
```

```
  }
}
```

Error Response Structure

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Error description",
  "action_time": "2025-10-02T10:30:45",
  "data": "Error description"
}
```

Standard Response Fields

Field	Type	Description
`success`	boolean	Always `true` for successful operations, `false` for errors
`httpStatus`	string	HTTP status name (OK, CREATED, BAD_REQUEST, NOT_FOUND, etc.)
`message`	string	Human-readable message describing the operation result
`action_time`	string	ISO 8601 timestamp of when the response was generated
`data`	object/string	Response payload for success, error details for failures

---

HTTP Method Badge Standards

For better visual clarity, all endpoints use colored badges for HTTP methods:

- **GET** - `GET` - Green (Safe, read-only operations)

Transaction Types

Type	Description	Direction

WALLET_TOPUP	Money added to wallet from external source	CREDIT
WALLET_WITHDRAWAL	Money withdrawn from wallet to external account	DEBIT
PURCHASE	Payment made for product purchase	DEBIT
PURCHASE_REFUND	Refund received for cancelled purchase	CREDIT
SALE	Earnings received from selling products	CREDIT
SALE_REFUND	Refund issued to buyer for sale	DEBIT
PLATFORM_FEE_COLLECTED	Platform fee collected (admin tracking)	CREDIT
GROUP_PURCHASE	Payment for group buying deal	DEBIT
GROUP_REFUND	Refund for failed group deal	CREDIT
INSTALLMENT_PAYMENT	One installment payment (future)	DEBIT
INSTALLMENT_REFUND	Refund for cancelled installment (future)	CREDIT
ESCROW_HOLD	Money held in escrow (admin tracking)	DEBIT
ESCROW_RELEASE	Money released from escrow (admin tracking)	CREDIT
ESCROW_REFUND	Money refunded from escrow	CREDIT

Transaction Directions

Direction	Description	Display
DEBIT	Money leaving wallet	Negative amount (-)
CREDIT	Money entering wallet	Positive amount (+)

Endpoints

1. Get My Transactions

Purpose: Retrieves all transactions for the authenticated user with pagination support. Returns transactions ordered by creation date (newest first).

Endpoint: `GET` `{base_url}`

Access Level: □□ Protected (Requires Authentication)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated user

Query Parameters:

Parameter	Type	Required	Description	Default
page	integer	No	Page number (0-indexed)	0
size	integer	No	Number of items per page	20

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Transactions retrieved successfully",
  "action_time": "2025-10-02T10:30:45",
  "data": {
    "content": [
      {
        "id": "t1r2a3n4-s5a6-7890-abcd-ef1234567890",
        "transactionRef": "#2025T000123",
        "type": "PURCHASE",
        "direction": "DEBIT",
        "amount": 150000.00,
        "displayAmount": -150000.00,
        "currency": "TZS",
        "title": "Purchase Payment",
        "description": "Payment for order (Escrow: ESC-2025-000045)",
        "status": "COMPLETED",
```

```
      "createdAt": "2025-10-02T10:15:30",
      "referenceType": "ESCROW",
      "referenceId": "e1s2c3r4-o5w6-7890-abcd-ef1234567890"
    },
    {
      "id": "t2r3a4n5-s6a7-8901-bcde-f12345678901",
      "transactionRef": "#2025T000122",
      "type": "WALLET_TOPUP",
      "direction": "CREDIT",
      "amount": 50000.00,
      "displayAmount": 50000.00,
      "currency": "TZS",
      "title": "Wallet Topup",
      "description": "M-Pesa top-up from +255712345678",
      "status": "COMPLETED",
      "createdAt": "2025-10-02T09:30:15",
      "referenceType": "WALLET",
      "referenceId": "w1a2l3l4-e5t6-7890-abcd-ef1234567890"
    }
  ],
  "pageable": {
    "pageNumber": 0,
    "pageSize": 20,
    "sort": {
      "sorted": true,
      "unsorted": false,
      "empty": false
    },
    "offset": 0,
    "paged": true,
    "unpaged": false
  },
  "totalElements": 47,
  "totalPages": 3,
  "last": false,
  "size": 20,
  "number": 0,
  "sort": {
    "sorted": true,
    "unsorted": false,
```

```
      "empty": false
    },
    "numberOfElements": 20,
    "first": true,
    "empty": false
  }
}
```

Success Response Fields:

Field	Description
id	Unique identifier for the transaction
transactionRef	Unique transaction reference number
type	Transaction type (see Transaction Types table)
direction	DEBIT or CREDIT
amount	Absolute transaction amount
displayAmount	Amount with sign (negative for DEBIT, positive for CREDIT)
currency	Currency code (TZS)
title	Short transaction title
description	Detailed transaction description
status	Transaction status (COMPLETED, PENDING, FAILED)
createdAt	ISO 8601 timestamp when transaction was created
referenceType	Type of related entity (WALLET, ESCROW, ORDER, etc.)
referenceId	UUID of related entity

Error Response Examples:

Unauthorized (401):

```
{
  "success": false,
  "httpStatus": "UNAUTHORIZED",
  "message": "Authentication token is required",
  "action_time": "2025-10-02T10:30:45",
  "data": "Authentication token is required"
}
```

---

2. Get Transaction by ID

Purpose: Retrieves detailed information about a specific transaction by its ID. Only the transaction owner can access their transactions.

Endpoint: `GET` `{base_url}/{id}`

Access Level: □□ Protected (Requires Authentication and Ownership)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated user

Path Parameters:

Parameter	Type	Required	Description	Validation
id	string (UUID)	Yes	Unique identifier of the transaction	Valid UUID format

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Transaction retrieved successfully",
  "action_time": "2025-10-02T10:35:45",
  "data": {
    "id": "t1r2a3n4-s5a6-7890-abcd-ef1234567890",
    "transactionRef": "#2025T000123",
    "type": "PURCHASE",
    "direction": "DEBIT",
    "amount": 150000.00,
    "displayAmount": -150000.00,
    "currency": "TZS",
    "title": "Purchase Payment",
    "description": "Payment for order (Escrow: ESC-2025-000045)",
    "status": "COMPLETED",
    "createdAt": "2025-10-02T10:15:30",
```

```
    "referenceType": "ESCROW",
    "referenceId": "e1s2c3r4-o5w6-7890-abcd-ef1234567890"
  }
}
```

Error Response Examples:

Not Found (404):

```
{
  "success": false,
  "httpStatus": "NOT_FOUND",
  "message": "Transaction not found",
  "action_time": "2025-10-02T10:35:45",
  "data": "Transaction not found"
}
```

---

3. Get Transaction by Reference

Purpose: Retrieves a transaction by its unique reference number (e.g., #2025T000123).

Endpoint: `GET` `{base_url}/ref/{transactionRef}`

Access Level: Protected (Requires Authentication and Ownership)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated user

Path Parameters:

Parameter	Type	Required	Description	Validation
transactionRef	string	Yes	Unique transaction reference number	Format: #YYYYTNNNNNN

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Transaction retrieved successfully",
  "action_time": "2025-10-02T10:40:45",
  "data": {
    "id": "t1r2a3n4-s5a6-7890-abcd-ef1234567890",
    "transactionRef": "#2025T000123",
    "type": "PURCHASE",
    "direction": "DEBIT",
    "amount": 150000.00,
    "displayAmount": -150000.00,
    "currency": "TZS",
    "title": "Purchase Payment",
    "description": "Payment for order (Escrow: ESC-2025-000045)",
    "status": "COMPLETED",
    "createdAt": "2025-10-02T10:15:30",
    "referenceType": "ESCROW",
    "referenceId": "e1s2c3r4-o5w6-7890-abcd-ef1234567890"
  }
}
```


Error Response Examples:

Not Found (404):


```
{
  "success": false,
  "httpStatus": "NOT_FOUND",
  "message": "Transaction not found: #2025T000123",
  "action_time": "2025-10-02T10:40:45",
  "data": "Transaction not found: #2025T000123"
}
```


---

4. Get Transactions by Type

Purpose: Retrieves all transactions of a specific type for the authenticated user with pagination support.

Endpoint: `GET` `{base_url}/filter/type`

Access Level: □□ Protected (Requires Authentication)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated user

Query Parameters:

Parameter	Type	Required	Description	Default
type	string (enum)	Yes	Transaction type to filter by	-
page	integer	No	Page number (0-indexed)	0
size	integer	No	Number of items per page	20

Valid Type Values:

- WALLET_TOPUP
- WALLET_WITHDRAWAL
- PURCHASE
- PURCHASE_REFUND
- SALE
- SALE_REFUND
- PLATFORM_FEE_COLLECTED
- GROUP_PURCHASE
- GROUP_REFUND
- INSTALLMENT_PAYMENT
- INSTALLMENT_REFUND
- ESCROW_HOLD
- ESCROW_RELEASE
- ESCROW_REFUND

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Transactions retrieved successfully",
```

```
  "action_time": "2025-10-02T10:45:45",
  "data": {
    "content": [
      {
        "id": "t1r2a3n4-s5a6-7890-abcd-ef1234567890",
        "transactionRef": "#2025T000123",
        "type": "PURCHASE",
        "direction": "DEBIT",
        "amount": 150000.00,
        "displayAmount": -150000.00,
        "currency": "TZS",
        "title": "Purchase Payment",
        "description": "Payment for order (Escrow: ESC-2025-000045)",
        "status": "COMPLETED",
        "createdAt": "2025-10-02T10:15:30",
        "referenceType": "ESCROW",
        "referenceId": "e1s2c3r4-o5w6-7890-abcd-ef1234567890"
      }
    ],
    "totalElements": 15,
    "totalPages": 1,
    "size": 20,
    "number": 0
  }
}
```


Error Response Examples:

Bad Request - Invalid Type (400):


```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Invalid transaction type",
  "action_time": "2025-10-02T10:45:45",
  "data": "Invalid transaction type"
}
```

5. Get Transactions by Direction

Purpose: Retrieves all transactions of a specific direction (DEBIT or CREDIT) for the authenticated user with pagination support.

Endpoint: `GET` `{base_url}/filter/direction`

Access Level: □□ Protected (Requires Authentication)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated user

Query Parameters:

Parameter	Type	Required	Description	Default
direction	string (enum)	Yes	Transaction direction (DEBIT or CREDIT)	-
page	integer	No	Page number (0-indexed)	0
size	integer	No	Number of items per page	20

Valid Direction Values:

- DEBIT (money out)
- CREDIT (money in)

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Transactions retrieved successfully",
  "action_time": "2025-10-02T10:50:45",
  "data": {
    "content": [
      {
        "id": "t2r3a4n5-s6a7-8901-bcde-f12345678901",
```

```
        "transactionRef": "#2025T000122",
        "type": "WALLET_TOPUP",
        "direction": "CREDIT",
        "amount": 50000.00,
        "displayAmount": 50000.00,
        "currency": "TZS",
        "title": "Wallet Topup",
        "description": "M-Pesa top-up from +255712345678",
        "status": "COMPLETED",
        "createdAt": "2025-10-02T09:30:15",
        "referenceType": "WALLET",
        "referenceId": "w1a2l3l4-e5t6-7890-abcd-ef1234567890"
      }
    ],
    "totalElements": 23,
    "totalPages": 2,
    "size": 20,
    "number": 0
  }
}
```


Error Response Examples:

Bad Request - Invalid Direction (400):


```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Invalid transaction direction",
  "action_time": "2025-10-02T10:50:45",
  "data": "Invalid transaction direction"
}
```


---

6. Get Transactions by Date Range

Purpose: Retrieves all transactions within a specified date range for the authenticated user with pagination support.

Endpoint: `GET` `{base_url}/filter/date-range`

Access Level: □□ Protected (Requires Authentication)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated user

Query Parameters:

Parameter	Type	Required	Description	Default
startDate	string (ISO 8601)	Yes	Start date and time (inclusive)	-
endDate	string (ISO 8601)	Yes	End date and time (inclusive)	-
page	integer	No	Page number (0-indexed)	0
size	integer	No	Number of items per page	20

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Transactions retrieved successfully",
  "action_time": "2025-10-02T10:55:45",
  "data": {
    "content": [
      {
        "id": "t1r2a3n4-s5a6-7890-abcd-ef1234567890",
        "transactionRef": "#2025T000123",
        "type": "PURCHASE",
        "direction": "DEBIT",
        "amount": 150000.00,
        "displayAmount": -150000.00,
        "currency": "TZS",
        "title": "Purchase Payment",
        "description": "Payment for order (Escrow: ESC-2025-000045)",
        "status": "COMPLETED",
```

```
        "createdAt": "2025-10-02T10:15:30",
        "referenceType": "ESCROW",
        "referenceId": "e1s2c3r4-o5w6-7890-abcd-ef1234567890"
      }
    ],
    "totalElements": 8,
    "totalPages": 1,
    "size": 20,
    "number": 0
  }
}
```


Error Response Examples:

Bad Request - Invalid Date Format (400):


```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Invalid date format. Use ISO 8601 format",
  "action_time": "2025-10-02T10:55:45",
  "data": "Invalid date format. Use ISO 8601 format"
}
```


---

7. Get Transaction Count

Purpose: Retrieves the total count of all transactions for the authenticated user.

Endpoint: `GET` `{base_url}/count`

Access Level: □□ Protected (Requires Authentication)

Authentication: Bearer Token required in Authorization header

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer token for authenticated user

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Transaction count retrieved successfully",
  "action_time": "2025-10-02T11:00:45",
  "data": 47
}
```

Error Response Examples:

Unauthorized (401):

```
{
  "success": false,
  "httpStatus": "UNAUTHORIZED",
  "message": "Authentication token is required",
  "action_time": "2025-10-02T11:00:45",
  "data": "Authentication token is required"
}
```

Integration Examples

Example 1: Get Recent Transactions

```
GET /api/v1/transaction-history?page=0&size=10
Authorization: Bearer {token}
```

Response: Returns 10 most recent transactions

Example 2: View All Purchases

```
GET /api/v1/transaction-history/filter/type?type=PURCHASE&page=0&size=20
Authorization: Bearer {token}
```

Response: Returns all purchase transactions

Example 3: Check Money Received (Credits)

```
GET /api/v1/transaction-history/filter/direction?direction=CREDIT&page=0&size=20
Authorization: Bearer {token}
```

Response: Returns all credit (incoming) transactions

Example 4: Get Monthly Statement

```
GET /api/v1/transaction-history/filter/date-range?startDate=2025-09-01T00:00:00Z&endDate=2025-09-30T23:59:59Z&page=0&size=100
Authorization: Bearer {token}
```

Response: Returns all transactions for September 2025

Example 5: Track Specific Transaction

Step 1: Get Transaction by Reference

```
GET /api/v1/transaction-history/ref/#2025T000123
Authorization: Bearer {token}
```

Step 2: Get Full Details by ID

```
GET /api/v1/transaction-history/t1r2a3n4-s5a6-7890-abcd-ef1234567890
Authorization: Bearer {token}
```

Rate Limiting

Rate Limits:

- All Endpoints: 60 requests per minute per user

Rate Limit Headers:

```
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 45
```

```
X-RateLimit-Reset: 1696258800
```

Rate Limit Exceeded:

```
{
  "success": false,
  "httpStatus": "TOO_MANY_REQUESTS",
  "message": "Rate limit exceeded. Please try again later.",
  "action_time": "2025-10-02T11:05:45",
  "data": "Rate limit exceeded. Please try again later."
}
```

Best Practices

For Developers

1. **Use pagination** for large result sets
2. **Cache transaction lists** with short TTL (1-2 minutes)
3. **Filter by type or direction** for specific views
4. **Use date ranges** for generating statements
5. **Display displayAmount** field for user-friendly amounts with signs
6. **Show transaction reference** for user tracking and support
7. **Link to related entities** using referenceType and referenceId

For UI/UX

1. **Color-code transactions**: Green for CREDIT, Red for DEBIT
2. **Group by date**: Show transactions by day/month
3. **Provide filters**: Type, direction, date range
4. **Show running balance**: Calculate balance after each transaction
5. **Enable search**: By reference number or description
6. **Export capability**: Allow CSV/PDF export of statements
7. **Infinite scroll or pagination**: For better mobile experience

---