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."
}
```

Revision #4
Created 2 October 2025 06:41:34 by Admin Qbit
Updated 21 May 2026 17:54:00 by Admin Qbit