Disbursement

- Disbursement Channel API
- Disbursement API

Disbursement Channel API

Author: Josh — Lead Backend Team **Last Updated**: 2026-03-06 **Version**: v1.0

Base URL: `https://api.nextgate.co.tz/api/v1`

Description: Manages withdrawal destinations for NextGate users. A channel is a verified mobile money or bank account to which a user can withdraw their wallet balance. Channels require OTP verification to add and have a cooling period before first use.

Key notes:

- Adding a channel is a **three-step flow**: `/lookup` → `/add` → `/add/confirm`
- The `/lookup` step returns a signed `confirmationToken` required by `/add` — expires in 10 minutes
- Selcom name lookup is performed at both `/lookup` and `/add` — verified twice server-side
- The **first channel** on an account activates immediately — 24-hour cooling applies to all subsequent channels
- Deleting a channel requires OTP confirmation — two-step: `DELETE /{channelId}` → `DELETE /{channelId}/confirm`
- Deleted channels cannot be used for new withdrawals — in-flight disbursements snapshot channel data at initiation time and are unaffected

---

Security & Client Guidelines

Authentication

All endpoints require a valid JWT Bearer token:

```
Authorization: Bearer <your_token>
```

On `401 UNAUTHORIZED`, silently refresh the token and retry once. If refresh fails, redirect to login.

Never store the JWT in localStorage — use secure device storage on mobile, or HttpOnly cookies on web.

OTP Token Handling

Both the add and delete flows are two-step. The first step returns an `otpToken`. Store this in memory and pass it to the confirm step along with the SMS OTP code. Do not log it or persist it beyond the confirmation flow.

The OTP expires after ~5 minutes. If the user is too slow, the confirm step returns an error — prompt them to **restart the entire flow from the beginning**. The server must clean up the previous pending channel record before a new one can be created.

If the wrong OTP is entered too many times, the OTP is locked and the flow is terminated. The user must start over.

Name Lookup and Confirmation Token

The `/lookup` step returns a signed `confirmationToken` that encodes the destination, channel type, and account ID. It is valid for **10 minutes**.

The client **must** pass this token to `/add`. The server rejects the request if the token is missing, expired, or does not match the destination and channel type in the request body.

Always display `accountHolderName` prominently to the user and require explicit confirmation before calling `/add`. This protects users from accidentally adding a wrong destination.

Cooling Period UX

After OTP confirmation, `isUsable` is `false` until `activatesAt`. Show a message like:

> *"This channel will be ready for withdrawals on {activatesAt date}"*

Visually distinguish usable channels from those still in the cooling period when listing.

Exception: The first channel added to an account activates immediately (`isUsable: true`, `activatesAt` set to now).

Deleting a Channel

Deletion is irreversible. Show a clear confirmation dialog before initiating the delete flow.

If the deleted channel was the primary channel, the server automatically promotes another active channel as primary.

Pending disbursements that reference a deleted channel are not affected — the disbursement holds a snapshot of the channel data from initiation time.

Network Error Handling

HTTP Status	Meaning	Client action
`400`	Business rule violation	Show `message` to user, do not retry
`401`	Token expired or invalid	Refresh token silently, retry once
`403`	Forbidden	Show error, do not retry
`422`	Validation error	Show field errors, fix and resubmit
`500`	Server error	Show generic error, allow manual retry
Network timeout	No response	Lookup and list: safe to retry. Initiate/confirm: check status first

---

Standard Response Format

Success

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Operation completed successfully",
  "action_time": "2026-03-06T10:30:45",
  "data": {}
}
```

Error

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Error description",
  "action_time": "2026-03-06T10:30:45",
  "data": "Error description"
```

```
}
```

Flow Diagram

```
ADD CHANNEL FLOW
----------------
User
 |
 |--- POST /disbursement/channels/lookup -------> Selcom Name Lookup
 |       { channelType, destination, bankCode? }          |
 |<-- { accountHolderName, destinationDisplay,    Returns verified name
 |       channelType, confirmationToken }
 |                                          <-- store confirmationToken (expires 10 min)
 |   [User reviews name, confirms]
 |
 |--- POST /disbursement/channels/add ----------> Validate confirmationToken
 |       { channelType, destination,               Re-verify name with Selcom
 |         bankCode?, confirmationToken }           Check for duplicate
 |                                                  Save channel record
 |                                                  Generate + send OTP
 |<-- { otpToken }
 |
 |--- POST /disbursement/channels/add/confirm --> Verify OTP
 |       ?otpToken=...&otpCode=...                 Activate channel
 |<-- { channelId, isUsable, activatesAt, ... }    (cooling 24h, except first channel)
 |
 |   [Channel becomes usable after activatesAt]
 |
 |--- GET /disbursement/channels ---------------> List active channels
 |<-- [ { channelId, isUsable, activatesAt, ... } ]



DELETE CHANNEL FLOW
-------------------
User
 |
 |--- DELETE /disbursement/channels/{channelId} ---------> Generate + send OTP
 |<-- { otpToken }
```

```
|
|--- DELETE /disbursement/channels/{channelId}/confirm --> Verify OTP
|       ?otpToken=...&otpCode=...                          Soft delete channel
|<-- 200 OK                                                Reassign primary if needed
```

Channel Status Reference

Status	Description
`PENDING_ACTIVATION`	Channel OTP confirmed, within 24-hour cooling period — not yet usable
`ACTIVE`	Channel verified and activated
`DELETED`	Channel removed by user

> A cleanup job runs nightly and soft-deletes any channel records whose OTP was never verified (i.e. the add flow was abandoned before confirm).

Endpoints

---

1. Lookup Account Name

Purpose: Verifies a destination account and returns the registered account holder name from Selcom, along with a signed `confirmationToken` required for the next step. Always call this before `/add`.

Endpoint: `POST /disbursement/channels/lookup`

Access: □□ Protected — Bearer Token required

Request Body:

```
{
  "channelType": "MPESA",
  "destination": "255712345678",
```

```
  "bankCode": null
}
```

Parameter	Type	Required	Description
`channelType`	string	Yes	`MPESA`, `AIRTEL`, `TIGOPESA`, `HALOPESA`, `SELCOM_PESA`, `BANK`
`destination`	string	Yes	Phone number (`255XXXXXXXXX`) or bank account number
`bankCode`	string	Conditional	Required when `channelType` is `BANK`

Success Response:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Account verified successfully",
  "action_time": "2026-03-06T10:30:45",
  "data": {
    "accountHolderName": "JOHN DOE",
    "destinationDisplay": "2557****678",
    "channelType": "MPESA",
    "confirmationToken": "<signed-token>"
  }
}
```

Field	Description
`accountHolderName`	Verified name from Selcom — display to user for confirmation
`destinationDisplay`	Masked destination
`channelType`	The channel type
`confirmationToken`	Signed token required by `/add`. Valid for 10 minutes. Store in memory

Errors:

Scenario	Message
Account not found	`"Account not found. Please check the number and try again."`

Scenario	Message
Phone not verified	`"Your phone number must be verified before adding a withdrawal channel."`
Destination already added	`"This destination is already added as a withdrawal channel."`
Bank code missing	`"Bank code is required for bank channels."`
Selcom lookup failed	`"Could not verify account. Please check the details and try again."`

---

2. Initiate Add Channel

Purpose: Initiates adding a new channel. Validates the `confirmationToken` from `/lookup`, re-verifies the name with Selcom server-side, saves the channel record, and sends an OTP to the user's verified phone.

Endpoint: `POST /disbursement/channels/add`

Access: □□ Protected — Bearer Token required

Request Body (Mobile Money):

```
{
  "channelType": "MPESA",
  "destination": "255712345678",
  "bankCode": null,
  "confirmationToken": "<token from /lookup>"
}
```

Request Body (Bank):

```
{
  "channelType": "BANK",
  "destination": "0012345678901",
  "bankCode": "CRDB",
  "confirmationToken": "<token from /lookup>"
}
```

Parameter	Type	Required	Description
`channelType`	string	Yes	`MPESA`, `AIRTEL`, `TIGOPESA`, `HALOPESA`, `SELCOM_PESA`, `BANK`

Parameter	Type	Required	Description
`destination`	string	Yes	Phone or account number
`bankCode`	string	Conditional	Required when `channelType` is `BANK`
`confirmationToken`	string	Yes	Token from `/lookup`. Must match destination, channel type, and account

Success Response:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "OTP sent to your verified phone number",
  "action_time": "2026-03-06T10:30:45",
  "data": {
    "otpToken": "otp-token-uuid-here"
  }
}
```

Field	Description
`otpToken`	Pass to `/add/confirm` with the SMS OTP code

Errors:

Scenario	Message
Token expired	`"Confirmation token expired. Please look up the account again."`
Token invalid/mismatch	`"Invalid confirmation token."`
Channel already active	`"This destination is already an active withdrawal channel."`
Max channels reached	`"Maximum of {n} withdrawal channels allowed."`
Add window closed	`"Withdrawal channel add window has closed."`

---

3. Confirm Add Channel

Purpose: Confirms adding a new channel by verifying the OTP. On success the channel is activated with a cooling period before first use (first channel activates immediately).

Endpoint: `POST /disbursement/channels/add/confirm`

Access: 🔒 Protected — Bearer Token required

Query Parameters:

Parameter	Type	Required	Description
`otpToken`	string	Yes	Token returned from `/add`
`otpCode`	string	Yes	6-digit OTP received via SMS

Success Response:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Channel added successfully",
  "action_time": "2026-03-06T10:30:45",
  "data": {
    "channelId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "channelType": "MPESA",
    "destinationDisplay": "2557****678",
    "accountHolderName": "JOHN DOE",
    "bankName": null,
    "isPrimary": false,
    "status": "ACTIVE",
    "isUsable": false,
    "activatesAt": "2026-03-07T10:31:00"
  }
}
```


Field	Description
`channelId`	UUID — use this when initiating withdrawals
`channelType`	Channel type
`destinationDisplay`	Masked destination
`accountHolderName`	Verified name from Selcom
`bankName`	Bank name — only for `BANK` channels, null otherwise
`isPrimary`	True if this is now the user's primary channel
`status`	`ACTIVE` or `PENDING_ACTIVATION`

Field	Description
`isUsable`	`false` if still in cooling period
`activatesAt`	When the channel becomes usable for withdrawals

Errors:

Scenario	Message
Invalid OTP	`"Invalid OTP code."`
OTP locked	`"OTP locked — max attempts exceeded."`
Channel no longer valid	`"This channel is no longer valid. Please add it again."`

---

4. List My Channels

Purpose: Returns all active withdrawal channels for the authenticated user.

Endpoint: `GET /disbursement/channels`

Access: 🔒 Protected — Bearer Token required

> Only `ACTIVE` channels are returned. Deleted and pending-activation channels that have not yet been OTP-confirmed are excluded.

Success Response:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Channels retrieved successfully",
  "action_time": "2026-03-06T10:30:45",
  "data": [
    {
      "channelId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
      "channelType": "MPESA",
      "destinationDisplay": "2557****678",
      "accountHolderName": "JOHN DOE",
      "bankName": null,
      "isPrimary": true,
```

```
      "status": "ACTIVE",
      "isUsable": true,
      "activatesAt": "2026-03-05T10:31:00"
    },
    {
      "channelId": "b2c3d4e5-f6a7-8901-bcde-f12345678901",
      "channelType": "BANK",
      "destinationDisplay": "0012****789",
      "accountHolderName": "JOHN DOE",
      "bankName": "CRDB Bank",
      "isPrimary": false,
      "status": "ACTIVE",
      "isUsable": false,
      "activatesAt": "2026-03-07T08:00:00"
    }
  ]
}
```


Field	Description
`channelId`	UUID of the channel
`channelType`	Channel type
`destinationDisplay`	Masked destination
`accountHolderName`	Verified name from Selcom
`bankName`	Bank name — only for `BANK` channels, null otherwise
`isPrimary`	Whether this is the user's primary channel
`status`	`ACTIVE`
`isUsable`	`true` if cooling period has passed and channel can be used now
`activatesAt`	When the cooling period ends

---

5. Initiate Delete Channel

Purpose: Initiates deletion of a withdrawal channel. Sends OTP to user's verified phone for confirmation.

Endpoint: `DELETE /disbursement/channels/{channelId}`

Access: 🔒 Protected — Bearer Token required

Path Parameters:

Parameter	Type	Required	Description
`channelId`	UUID	Yes	ID of the channel to delete

Success Response:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "OTP sent to your verified phone number",
  "action_time": "2026-03-06T10:30:45",
  "data": {
    "otpToken": "otp-token-uuid-here"
  }
}
```

Errors:

Scenario	Message
Channel not found / not owned	`"Channel not found."`
Channel not active	`"Only active channels can be deleted."`
Phone not verified	`"Your phone number must be verified to delete a withdrawal channel."`

---

6. Confirm Delete Channel

Purpose: Confirms channel deletion by verifying the OTP. Channel is permanently removed. If it was the primary channel, another active channel is automatically promoted.

Endpoint: `DELETE /disbursement/channels/{channelId}/confirm`

Access: 🔒 Protected — Bearer Token required

Path Parameters:

Parameter	Type	Required	Description
`channelId`	UUID	Yes	ID of the channel to delete

Query Parameters:

Parameter	Type	Required	Description
`otpToken`	string	Yes	Token returned from initiate delete
`otpCode`	string	Yes	6-digit OTP received via SMS

Success Response:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Channel deleted successfully",
  "action_time": "2026-03-06T10:30:45",
  "data": null
}
```

Errors:

Scenario	Message
Invalid OTP	`"Invalid OTP code."`
OTP does not match channel	`"OTP does not match this channel."`
Channel not active	`"Only active channels can be deleted."`

Quick Reference

Method	Endpoint	Description
`POST`	`/disbursement/channels/lookup`	Verify account name, get `confirmationToken`
`POST`	`/disbursement/channels/add`	Start add channel flow (requires `confirmationToken`)
`POST`	`/disbursement/channels/add/confirm`	Complete add channel with OTP
`GET`	`/disbursement/channels`	List all active user channels
`DELETE`	`/disbursement/channels/{channelId}`	Start delete channel flow
`DELETE`	`/disbursement/channels/{channelId}/confirm`	Complete delete with OTP

Disbursement API

Author: Josh Lead Backend Team
Last Updated: 2026-03-06
Version: v1.0

Base URL: `https://api.nextgate.co.tz/api/v1`

Short Description: The Disbursement API handles wallet withdrawals for NextGate users. Users can withdraw their wallet balance to a verified mobile money account or bank account. Each withdrawal requires OTP confirmation and deducts a platform fee plus Selcom transfer fee on top of the requested amount.

Hints:

- The withdrawal channel must be added and fully active (cooling period passed) before use — see Disbursement Channel API
- Total wallet debit = `requestedAmount + platformFee + selcomFee`
- The recipient receives exactly `requestedAmount` — fees are charged on top
- If Selcom returns `INPROGRESS`, the polling job resolves the final status automatically every 3 minutes — no frontend action needed
- Always provide a unique `idempotencyKey` per withdrawal request
- Poll `/status/{id}` after confirming to track final delivery

Security & Client Guidelines

Authentication

All endpoints in this API are protected. The client must include a valid JWT Bearer token in every request header:

```
Authorization: Bearer <your_token>
```

The token is obtained from the NextGate authentication API after login. When a request returns `401 UNAUTHORIZED`, the client must silently refresh the token and retry the original request once. If the refresh also fails, redirect the user to the login screen.

Never store the JWT token in a place accessible to third-party scripts. On mobile, use secure device storage. On web, prefer memory or HttpOnly cookies over localStorage.

Idempotency Key

The `idempotencyKey` field protects against duplicate withdrawals caused by network retries. This is critical for financial operations — without it, a network timeout could cause the user's wallet to be debited twice if the app retries blindly.

How to generate and use it correctly:

Generate the key exactly once at the moment the user initiates the withdrawal action — for example, when they tap the "Withdraw" button. Store this key in memory for the duration of that withdrawal flow. If the `/initiate` request fails due to a network error and your app retries, send the exact same key. The server will detect the duplicate and return the existing request rather than creating a new one.

Never generate a new key on each retry attempt. Never reuse a key from a previous completed or failed withdrawal. A new key must be generated fresh for each new withdrawal the user starts.

The key only protects `/initiate`. The `/confirm` step is protected by the `otpToken` and the pessimistic database lock on the disbursement record — a duplicate confirm call on the same request will be rejected because the status will no longer be `PENDING_OTP`.

OTP Token Handling

After calling `/initiate`, you receive an `otpToken`. This token must be held in memory and passed to `/confirm` along with the OTP code the user receives via SMS. Do not ask the user to copy or type the token — it is handled entirely by the app.

The OTP code expires after a short window (typically 5 minutes). If the user does not confirm in time, the confirm step will return an error and the withdrawal request will remain in `PENDING_OTP` status — no money has moved at this point. The user can restart the flow with a new idempotency key.

If the user enters the wrong OTP code too many times, the OTP is locked. The disbursement will be marked `FAILED` and the flow is terminated. No money has moved because the wallet is only debited after OTP verification. The user must start a new withdrawal request with a fresh idempotency key.

What Confirmation Actually Means

A `200 OK` response from `/confirm` means the OTP was verified, the wallet was debited, and the Selcom API was called. It does **not** mean the money has arrived at the recipient. Selcom processes

transfers asynchronously and may take seconds to minutes to confirm delivery.

Always redirect the user to a status screen after confirm and poll `/status/{id}` to show the final outcome. Do not show a "withdrawal successful" message based solely on the confirm response.

Status Polling After Confirm

After a successful `/confirm`, begin polling `/status/{id}` to track delivery.

Recommended polling behavior:

Poll every 5 seconds for the first minute. If still not in a terminal status, slow down to every 15 seconds for the next 5 minutes. After 6 minutes total with no terminal status, show the user a message explaining that the transfer is being processed and may take a little longer — this is the `AWAITING_CONFIRMATION` state where the server polling job is working in the background every 3 minutes.

Stop polling when status reaches any of these terminal states: `COMPLETED`, `REFUNDED`, `FAILED`, `MANUAL_REVIEW`.

Handling Each Terminal Status

Status	What to show the user
`COMPLETED`	Success — show amount sent, recipient name, transaction reference
`REFUNDED`	Transfer failed — show failure reason, confirm wallet has been refunded, suggest they try again
`FAILED`	OTP was locked — no money moved, suggest they start a new withdrawal
`MANUAL_REVIEW`	Show a message that the transfer is under review and they will be notified. Do not show this as a failure or a success. Provide the `supportRef` for the user to reference if they contact support

---

Fee Transparency

Before the user confirms the withdrawal, always display the full fee breakdown clearly so they know exactly how much will leave their wallet. Never show only the `requestedAmount` without also showing the fees.

A clear display should show: amount to recipient, platform fee, transfer fee, and total to be deducted from wallet. This prevents confusion when the user checks their wallet balance after the withdrawal and sees more deducted than they expected.

Network Error Handling

HTTP Status	What it means	Client action
`400`	Invalid request or business rule violation	Show `message` to user, do not retry automatically
`401`	Token expired or invalid	Refresh token silently, retry once
`403`	Forbidden — permission issue	Show error, do not retry
`422`	Validation error	Show field errors to user, fix and resubmit
`500`	Server error	Show generic error, allow user to retry manually
Network timeout on `/initiate`	No response received	Retry with the same idempotency key — server handles duplicates safely
Network timeout on `/confirm`	No response received	Do not retry automatically — check `/status/{id}` first to see if the confirm was already processed before retrying
Network timeout on `/status`	No response received	Safe to retry on next poll interval

---

Standard Response Format

Success Response

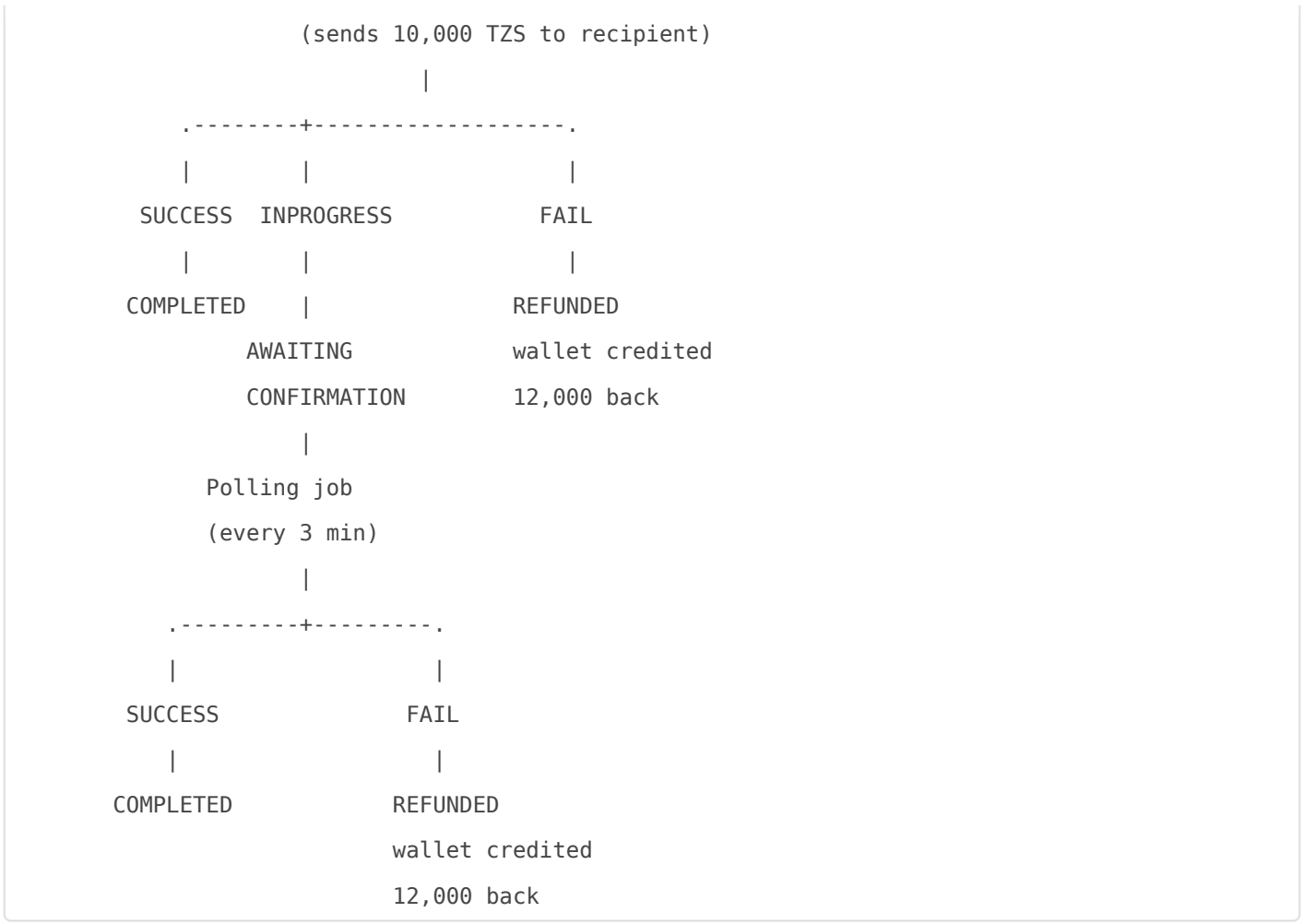
```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Operation completed successfully",
  "action_time": "2026-03-06T10:30:45",
  "data": {}
}
```

Error Response

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Error description",
  "action_time": "2026-03-06T10:30:45",
  "data": "Error description"
}
```

Money Flow Diagram

```
User requests withdrawal of 10,000 TZS
                |
        POST /disbursement/initiate
                |
        .-------+----------.
        |                  |
   Validate             Calculate fees
   channel              platformFee =  500 TZS
   balance              selcomFee   = 1,500 TZS
                        totalDebited = 12,000 TZS
                        disbursedAmt = 10,000 TZS
                             |
                  Save DisbursementRequest
                  Status: PENDING_OTP
                             |
                  Send OTP to verified phone
                             |
                  Return { otpToken }
                             |
        POST /disbursement/confirm
        (otpToken + otpCode)
                             |
                  Verify OTP
                             |
             Wallet debited 12,000 TZS
             Status: PROCESSING
                             |
                  Call Selcom API
```



Fee Structure

Component	Amount	Description
`requestedAmount`	User defined	What the recipient receives
`platformFee`	500 TZS	NextGate service charge
`selcomFee`	1,500 TZS	Selcom per-disbursement fee
`totalDebited`	requestedAmount + 2,000 TZS	Actual amount leaving wallet
`disbursedAmount`	= requestedAmount	Amount Selcom sends to recipient

Disbursement Status Reference

Status	Description
`PENDING_OTP`	Initiated, waiting for OTP confirmation — wallet not debited yet

Status	Description
`PROCESSING`	OTP confirmed, wallet debited, Selcom call in progress
`COMPLETED`	Selcom confirmed successful transfer
`AWAITING_CONFIRMATION`	Selcom returned INPROGRESS — polling job resolving every 3 min
`REFUNDED`	Selcom failed — wallet refunded full `totalDebited` automatically
`FAILED`	Failed due to OTP lock — no wallet debit occurred
`MANUAL_REVIEW`	Polling exhausted max retries — admin review required

Endpoints

1. Initiate Withdrawal

Purpose: Initiates a wallet withdrawal request. Validates channel, calculates fees, checks wallet balance, and sends OTP to user's verified phone number. No money moves at this step.

Endpoint: **POST** `/disbursement/initiate`

Access Level: □□ Protected — Requires Bearer Token

Authentication: `Authorization: Bearer <token>`

Request Headers:

Header	Type	Required	Description
Authorization	string	Yes	Bearer JWT token
Content-Type	string	Yes	`application/json`

Request JSON Sample:

```
{
  "channelId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "amount": 10000,
  "idempotencyKey": "usr-123-withdraw-1741234567"
}
```

Request Body Parameters:

Parameter	Type	Required	Description	Validation
`channelId`	UUID	Yes	ID of the verified withdrawal channel	Must belong to authenticated user and be active (cooling period passed)
`amount`	number	Yes	Amount to send to recipient in TZS	Min: 1000 TZS
`idempotencyKey`	string	Yes	Unique key to prevent duplicate withdrawals	Max 200 chars, unique per request

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "OTP sent to your verified phone number",
  "action_time": "2026-03-06T10:30:45",
  "data": {
  "otpToken": "dfe7e16f-cd8f-4c00-b0d0-8cff09201936"
}
}
```


Success Response Fields:

Field	Description
`data.otpToken`	OTP token string — pass this along with OTP code to `/confirm`

Error Responses:

Insufficient balance (400):


```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Insufficient balance. You need 12000 TZS (10000 + 500 platform fee + 1500
transfer fee).",
  "action_time": "2026-03-06T10:30:45",
  "data": "Insufficient balance. You need 12000 TZS (10000 + 500 platform fee + 1500 transfer
```

```
fee)."
}
```

Channel still in cooling period (400):

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "This withdrawal channel is not yet active.",
  "action_time": "2026-03-06T10:30:45",
  "data": "This withdrawal channel is not yet active."
}
```

Phone not verified (400):

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Your phone number must be verified before withdrawing.",
  "action_time": "2026-03-06T10:30:45",
  "data": "Your phone number must be verified before withdrawing."
}
```

Duplicate request (400):

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Duplicate request — this withdrawal is already being processed.",
  "action_time": "2026-03-06T10:30:45",
  "data": "Duplicate request — this withdrawal is already being processed."
}
```

Amount below minimum (400):

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Minimum withdrawal amount is 1000 TZS.",
  "action_time": "2026-03-06T10:30:45",
  "data": "Minimum withdrawal amount is 1000 TZS."
```

```
}
```

---

2. Confirm Withdrawal

Purpose: Confirms the withdrawal by verifying the OTP. Debits the wallet and calls Selcom to initiate the transfer. This is when money actually leaves the wallet.

Endpoint: `POST` `/disbursement/confirm`

Access Level: □□ Protected — Requires Bearer Token

Authentication: `Authorization: Bearer <token>`

Query Parameters:

Parameter	Type	Required	Description
`otpToken`	string	Yes	Token returned from `/initiate`
`otpCode`	string	Yes	6-digit OTP received via SMS

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Withdrawal processed successfully",
  "action_time": "2026-03-06T10:30:45",
  "data": null
}
```

> **Important**: A `200 OK` here means the Selcom call was made. It does **not** guarantee the money has arrived at the recipient yet. Poll `/status/{id}` to confirm final delivery status.

Error Responses:

Invalid OTP (400):

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Invalid OTP code.",
  "action_time": "2026-03-06T10:30:45",
  "data": "Invalid OTP code."
}
```

OTP locked — max attempts exceeded (400):

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "OTP locked — max attempts exceeded.",
  "action_time": "2026-03-06T10:30:45",
  "data": "OTP locked — max attempts exceeded."
}
```

Already processing (400):

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "This withdrawal is already being processed.",
  "action_time": "2026-03-06T10:30:45",
  "data": "This withdrawal is already being processed."
}
```

---

3. Get Disbursement Status

Purpose: Returns the current status and full details of a withdrawal request. Poll this after confirm to track final delivery.

Endpoint: `GET` `/disbursement/status/{disbursementRequestId}`

Access Level: □□ Protected — Requires Bearer Token

Authentication: `Authorization: Bearer <token>`

Path Parameters:

Parameter	Type	Required	Description
`disbursementRequestId`	UUID	Yes	ID of the disbursement request

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Disbursement status retrieved",
  "action_time": "2026-03-06T10:30:45",
  "data": {
    "disbursementRequestId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "requestedAmount": 10000,
    "platformFee": 500,
    "selcomFee": 1500,
    "totalDebited": 12000,
    "disbursedAmount": 10000,
    "currency": "TZS",
    "destination": "255712****78",
    "accountHolderName": "JOHN DOE",
    "status": "COMPLETED",
    "failureReason": null,
    "transactionRef": "NXT-TXN-20260306-ABCD1234",
    "supportRef": null,
    "createdAt": "2026-03-06T10:30:00",
    "completedAt": "2026-03-06T10:31:45"
  }
}
```


Success Response Fields:

Field	Description
`disbursementRequestId`	UUID of this withdrawal request
`requestedAmount`	Amount sent to recipient in TZS
`platformFee`	NextGate fee charged on top
`selcomFee`	Selcom transfer fee charged on top
`totalDebited`	Total amount debited from wallet
`disbursedAmount`	Amount Selcom sent to recipient — equals `requestedAmount`

Field	Description
`currency`	Always `TZS`
`destination`	Masked destination e.g. `255712****78`
`accountHolderName`	Verified recipient name
`status`	Current status — see status table above
`failureReason`	Populated if `FAILED` or `REFUNDED`
`transactionRef`	Wallet transaction reference
`supportRef`	Support reference for escalations — populated on `MANUAL_REVIEW`
`createdAt`	When withdrawal was initiated
`completedAt`	When transfer was confirmed — null if not yet completed

Error Responses:

Not found or not owned by user (400):

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Disbursement request not found",
  "action_time": "2026-03-06T10:30:45",
  "data": "Disbursement request not found"
}
```

Quick Reference

Method	Endpoint	Description
POST	`/disbursement/initiate`	Start withdrawal, sends OTP
POST	`/disbursement/confirm`	Confirm with OTP, debits wallet, calls Selcom
GET	`/disbursement/status/{id}`	Check withdrawal status