Calls & Meetings API Documentation

Author: Josh S. Sakweli, Backend Lead Team **Last Updated**: 2026-08-20 **Version**: v1.0

Base URL: `http://localhost:8765/api/v1` (local) — `server.port=8765`

Short Description: Real-time voice and video inside the chat system. Two surfaces sit here: **Calls** , which ring somebody now, and **Meetings**, which are scheduled or open rooms people join. Both are signalling only — this API issues room credentials and records what happened; the media itself never touches the backend, it goes phone → LiveKit SFU → phone.

Hints:

- Every endpoint needs `Authorization: Bearer <token>`.
- A join token is for **one room, one identity, and expires in 10 minutes** (`app.livekit.token-ttl-seconds=600`). Fetch it when you are about to connect, not when you render a list.
- Refusals are `409` with a machine-readable `code`. Branch on the code, never on the prose.
- The server owns call state and learns it from **LiveKit webhooks**, not from clients. A phone that dies mid-call never tells anyone it left.
- An unanswered call becomes `MISSED` after **45 seconds** (`app.livekit.ring-timeout-seconds=45`).

Standard Response Format

Success responses use the Globe Response Builder envelope:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Call started",
  "action_time": "2026-08-20T10:30:45",
  "data": { }
}
```

Refusals — read this before writing a client

A business refusal is **HTTP 409 CONFLICT**, and the envelope's `message` carries the **code**, with the code repeated inside `data`:

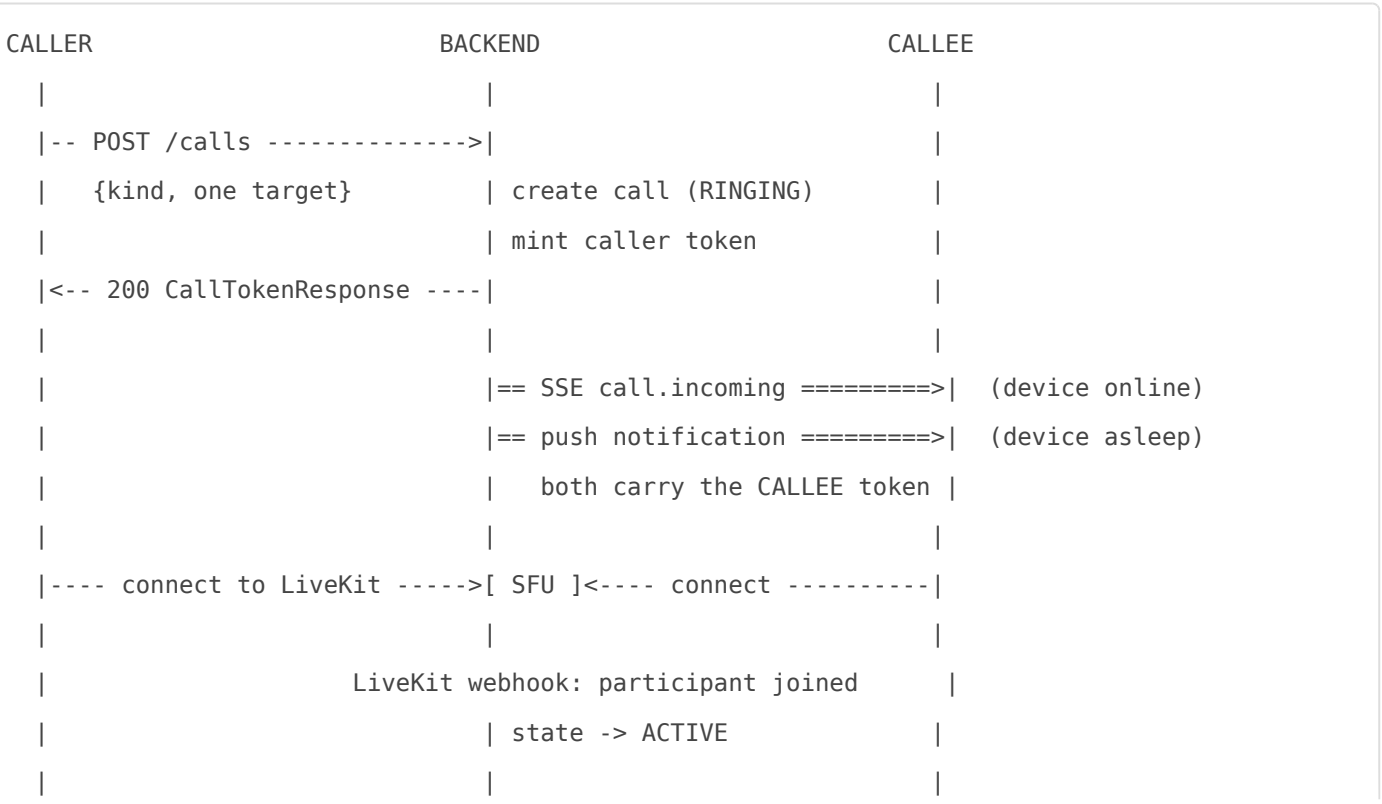
```
{
  "success": true,
  "httpStatus": "CONFLICT",
  "message": "BUSY",
  "action_time": "2026-08-20T10:30:45",
  "data": { "code": "BUSY", "reason": "That person is already in a call" }
}
```

Note `success` stays `true` — the envelope reports transport, not business outcome. **Treat any 409 as a refusal and read `data.code`.**

Part 1 — Calls

`api/v1/calls` — invite, accept and end are REST; the incoming alert is SSE plus push; media never comes near this controller.

The shape of a call

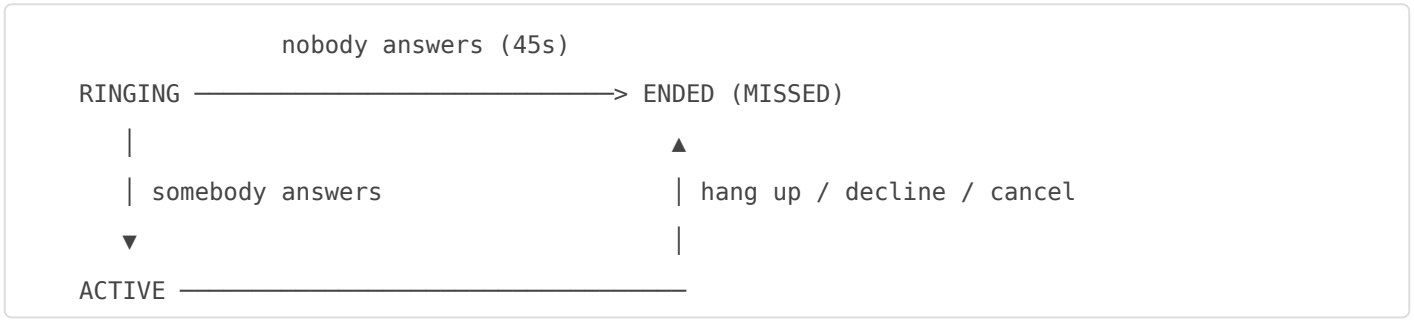


```
|-- POST /{id}/end ----------->| outcome COMPLETED           |
|                              |== SSE call.ended ============>|
|                              | writes CALL_EVENT to thread  |
```

Two things worth saying out loud in a demo:

1. The `call.incoming` event **carries the callee's token in the payload**. A ringing phone that had to fetch one first would add a round trip to answer latency. `POST /{callId}/join` exists only for a callee that learned about the call some other way, or cold-started.
2. A call always ends up attached to a conversation — that is where its `CALL_EVENT` message goes and what the permission check reads — but the caller never has to create one first.

Call state machine



`CallState` only ever advances. States: `RINGING`, `ACTIVE`, `ENDED` (plus `SCHEDULED` and `OPEN`, which belong to meetings).

`CallOutcome` is set once, on entry to `ENDED`, and copied into the thread's `CALL_EVENT`:

Outcome	Meaning
`COMPLETED`	reached `ACTIVE` — somebody answered
`MISSED`	rang out past the 45s timeout
`DECLINED`	callee explicitly refused
`CANCELLED`	caller hung up before it was answered
`FAILED`	could not be set up at all
`BUSY`	callee already in another call

`CANCELLED` and `MISSED` are deliberately separate — "I gave up" and "you never picked up" are different facts, and collapsing them loses the one the caller cares about.

`POST` /api/v1/calls — Start a call

Rings somebody. Returns the caller's own room credentials.

Request (`StartCallRequest`) — exactly **one** target must be present:

Field	Type	Required	Description
`conversationId`	UUID	one-of	Calling from an open thread
`targetShopId`	UUID	one-of	Calling a shop cold from its page; resolves or creates the commerce thread
`targetAccountId`	UUID	one-of	Calling a person you may not have messaged yet
`kind`	enum	**yes**	`AUDIO` or `VIDEO`
`shopId`	UUID	no	Present when calling **as** a shop (different from `targetShopId`). Validated against the caller's real contexts, never trusted

```
curl -s -X POST $BASE/api/v1/calls \
  -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
  -d '{"targetAccountId":"...","kind":"AUDIO"}'
```

Response (`CallTokenResponse`):

Field	Type	Description
`callId`	UUID	
`roomName`	string	LiveKit room
`token`	string	One room, one identity, 10-minute TTL
`url`	string	LiveKit websocket URL
`kind`	enum	`AUDIO` / `VIDEO`
`state`	enum	`RINGING`

`kind` is fixed for the call's lifetime. A video call that drops to audio is still a `VIDEO` call that degraded, and the **client** decides that, not the server.

`POST` /api/v1/calls/{callId}/join — Answer

For a callee that learned about the call over SSE or a cold start. A push-woken client already has its token in the payload and **skips this round trip entirely**.

Param	In	Required	Description
`callId`	path	yes	
`shopId`	query	no	Answering as a shop

Returns `CallTokenResponse`.

`POST` /api/v1/calls/{callId}/end — Hang up, decline, or cancel

All three are one verb. Which of the three it was is decided by the call's state and who asked — not by the client naming it — so a client cannot report a call it declined as one it completed.

Param	In	Required
`callId`	path	yes
`shopId`	query	no

Returns `CallResponse`.

`POST` /api/v1/calls/{callId}/participants — Pull somebody in

Adds a person to a call already running. **Initiator only, personal calls only.**

Param	In	Required
`callId`	path	yes
`accountId`	query	**yes**

Refuses with `CALL_TOO_LARGE` past `app.livekit.group-max-participants` — a hard refusal rather than ringing the first N, because silently dropping members of a group call is worse than saying no:

nobody can tell who was left out.

`GET` /api/v1/calls — Call history

The Calls list. **Keyed on who was ON the call** rather than on conversations, so a call somebody added you to appears here even though its thread is not one of yours.

Param	In	Default
`shopId`	query	— (personal)
`limit`	query	`30`

Returns `CallResponse[]`.

`GET` /api/v1/calls/{callId} — One call

Returns `CallResponse`. **Carries no token** — a client that wants to join asks for one, so a stale poll cannot hand out live credentials.

`CallResponse`

Field	Type
`callId`	UUID
`conversationId`	UUID
`kind`	`AUDIO` / `VIDEO`
`state`	`RINGING` / `ACTIVE` / `ENDED`
`outcome`	see table above
`initiatorJid`	string
`createdAt`, `answeredAt`, `endedAt`	ISO-8601
`durationSeconds`	long

Call refusal codes

Code	When

`AMBIGUOUS_TARGET`	zero or more than one target given
`NO_COUNTERPARTY`	nobody on the other end to ring
`BUSY`	callee already in another call
`ALREADY_IN_CALL`	the caller is
`BLOCKED`	blocked in either direction
`CALL_TOO_LARGE`	past the participant cap
`CALL_ENDED`	joining or acting on a finished call
`NOT_A_PARTICIPANT`	you were never on it
`NOT_INITIATOR`	only the initiator may add participants
`NOT_A_PERSONAL_CALL`	add-participant is personal-calls-only
`NOT_INVITED`	not on the guest list

Part 2 — Meetings

`api/v1/meetings` — scheduled and open calls. Separate from calls because **the verbs differ**: you schedule and join a meeting; you ring and answer a call.

A meeting is a `CallEntity` too, so it shares the state machine — but it is joined rather than answered, so it **never rings and can never be `MISSED`**.

```
    SCHEDULED ──(its window opens)──> OPEN ──(host ends it)──> ENDED
```

`OPEN` means the room is live for its window — and **stays open when empty**, since somebody looking in at 16:58 must not end the 17:00 meeting.

Meeting flow

```
HOST                                BACKEND                    INVITEES
 |                                     |                          |
 |-- POST /meetings ------------------>| state SCHEDULED           |
 |   {title, scheduledAt, invitees}    |                          |
 |                                     |== SSE meeting.scheduled ==>|  (once, for the
calendar)
 |                                     |                          |
 |                        [ MeetingScheduleJob: window opens ]     |
```

```
  |                                  | state OPEN                |
  |                                  |== SSE meeting.starting ===>|  (NO token — see below)
  |                                  |                           |
  |-- POST /{id}/join ---------------->|<---- POST /{id}/join ------|
  |<-- CallTokenResponse -------------- |----> CallTokenResponse --->|
  |                                  |                           |
  |-- POST /{id}/end ----------------->| state ENDED               |
```

`meeting.starting` **carries no token**, unlike `call.incoming`. A meeting is joined when the user decides to, not by a phone deciding for them.

Roles — a bandwidth decision, not ceremony

Role	May
`HOST`	speak, promote, demote, end
`SPEAKER`	publish; shares the publisher cap
`LISTENER`	subscribe only

A speaker costs an uplink and one decode on every other device; a listener costs one downstream stream and no uplink at all. That is why the participant cap counts **speakers** and lets listeners run far higher — and why the role is **enforced in the token grant** rather than asked of the client.

Access

`CallAccess`	Meaning
`INVITED`	a guest list — behaves like a group call arranged in advance (default)
`OPEN`	anyone holding a join link — but still a NexGate account, because a participant without a JID is unaddressable and unblockable

---

`POST` /api/v1/meetings — Schedule a meeting

Request (`CreateMeetingRequest`):

Field	Type	Required	Description
`title`	string	**yes**	A call needs no name because nobody labels "phoning Asha"; a meeting does, because everybody labels "Friday pricing review"
`description`	string	no	
`scheduledAt`	ISO-8601	no	Absent means now
`endsAt`	ISO-8601	no	When the room closes itself. Approximate by nature — not a promise about when people stop talking
`kind`	enum	no	`AUDIO` / `VIDEO`
`access`	enum	no	`INVITED` (default) or `OPEN`
`conversationId`	UUID	no	Anchored to a thread → writes its `CALL_EVENT` there; standalone → recorded in `GET /api/v1/calls` instead
`inviteeAccountIds`	UUID[]	no	

Returns `CallResponse` with `state: SCHEDULED`.

`POST` /api/v1/meetings/{meetingId}/join — Join

Returns `CallTokenResponse`. The token's grants encode the caller's role.

`POST` /api/v1/meetings/{meetingId}/links — Create a join link

Role-scoped: a speaker link and a listener link are different links.

Param	In	Default

`meetingId`	path	—
`role`	query	`LISTENER`

Response:

```
{ "token": "…", "role": "LISTENER", "expiresAt": "2026-09-19T…" }
```

Links live **30 days**. You cannot mint a `HOST` link — that refuses with `CANNOT_SHARE_HOST`.

`POST` /api/v1/meetings/join/{token} — Join by link

Path takes the link token. Still requires an authenticated NexGate account. Returns `CallTokenResponse`.

`PUT` /api/v1/meetings/{meetingId}/participants/role — Promote / demote

Param	In	Required
`jid`	query	yes
`role`	query	yes (`HOST` / `SPEAKER` / `LISTENER`)

Host only.

`POST` /api/v1/meetings/{meetingId}/end — End it

Returns `CallResponse` with `state: ENDED`.

Meeting refusal codes

Code	When
`TITLE_REQUIRED`	no title
`NOT_A_MEETING`	the id is a call, not a meeting
`NOT_HOST`	host-only verb
`NOT_INVITED`	`INVITED` access and you are not on the list
`MEETING_ENDED`	joining a finished meeting
`LINK_EXPIRED`	past the 30-day link lifetime
`SPEAKER_LIMIT`	stage is full; join as a listener
`CANNOT_SHARE_HOST`	you may not mint a HOST link
`BLOCKED`	blocked in either direction

Part 3 — The real-time channel

`GET` /api/v1/events (SSE)

One stream covers **every context the caller may act for**, so the context switcher and the permission check read the same source and cannot disagree.

Client note: the browser `EventSource` API cannot set an `Authorization` header, so **web must use fetch-based streaming**, not `EventSource`. Putting the token in the query string would leak credentials into access logs and proxy history — a poor trade for a few lines of client code.

Resume with `Last-Event-ID` (header wins) or `?lastEventId=`.

Call and meeting events

Event	Carries a token?	Payload
`call.incoming`	**yes**	`callId`, `conversationId`, `kind`, `from`, `roomName`, `url`, `token`
`call.updated`	no	
`call.ended`	no	`callId`, `conversationId`, `outcome`, `durationSeconds`
`meeting.scheduled`	no	`meetingId`, `title`, `scheduledAt`, `hostJid`, `conversationId?`
`meeting.starting`	**no**, on purpose	

`call.ended` goes to **everyone, including whoever hung up** — their other devices are still ringing otherwise.

Signalling fires `AFTER_COMMIT` throughout: alerting someone about a call whose row rolled back would ring a phone for a call that does not exist. SSE reaches a device that already holds a connection; push wakes one that does not. **Both go out** — a device with a live stream ignores the duplicate, and guessing wrong means a call nobody hears.

`POST` /api/v1/calls/livekit/webhook — internal

LiveKit → backend. **This is what actually moves a call to `ACTIVE` and to `ENDED`.** Not for clients; signed by LiveKit.

Configuration

Property	Default	Meaning
`app.livekit.url`	`ws://127.0.0.1:7880`	SFU websocket
`app.livekit.token-ttl-seconds`	`600`	Join token lifetime
`app.livekit.ring-timeout-seconds`	`45`	Unanswered → `MISSED`
`app.livekit.group-max-participants`	`8`	Group call / meeting speaker cap

The `CallTimeoutJob` runs every 15 seconds against the 45-second timeout, so a missed call settles within a minute. The client stops its own ringing at 45s regardless.

Revision #1
Created 13 July 2026 06:32:33 by Admin Qbit
Updated 20 August 2026 16:49:07 by Admin Qbit