OpenEvent Workflow v3 – Master Text Specification (Lindy-Integrated)

(Part 1 – System Context Overview & Global Workflow Conventions)

System Context Overview

This document defines the full, executable-level workflow logic for the **OpenEvent AI-Powered Event Management Platform**.
It consolidates the Lindy-style event workflow (as used in *Workflow v3.pdf* and *Workflow Instance on Lindy.pdf*) into a single human- and machine-readable specification.
All wording, structure, and conditional logic follow the same semantics used in the backend workflow engine.
The text is designed so that an LLM-based controller (Codex, Lindy, or equivalent) can run the process directly without needing to reference external diagrams or images.

Actors

Actor	Role
LLM	Core reasoning and orchestration agent. Parses messages, decides next actions,
HIL (Human in the Loop)	Venue manager or operator who validates, edits, and approves LLM-drafted content before client delivery.
Client	External event requester interacting through chat or email.
Trigger	Automation agent executing background actions such as creating calendar events, scheduling site visits, reminders, or updating statuses.
DB (Database	Central persistence system storing all event metadata, workflow states, messages,

Core Workflow Variables

Variable	Description
`chosen_dat`	The date selected for the event and confirmed by the client
`locked_roo`	Identifier of the specific room chosen or confirmed for the event. `null` if not yet
`requiremen ts`	Structured object containing: `number_of_participants`, `seating_layout`, `event_duration` (start/end), and
`requiremen ts_hash`	Hash or signature of the above requirements as they were when last validated. Used to detect changes.
`room_eval_`	Hash of the room evaluation snapshot (date + requirements). Used to prevent
`selected_p roducts`	List of additional items chosen by the client (catering packages, beverages, add-ons, technical services, etc.).
`caller_ste`	Reference to the step that initiated a detour, ensuring correct return flow after

Detour and Return Principle

1. Whenever a step encounters missing or changed data (e.g. new date, new room, changed requirements), it triggers a **detour** to the specific step responsible for resolving that data.

2. Before jumping, the system sets `caller_step = <current step>`.

3. Once the resolving step finishes, control **returns to `caller_step`**, unless the entry-guard of that step redirects elsewhere (e.g. to re-evaluate rooms after a date change).

4. Only dependent steps re-run; unaffected steps remain cached.

This ensures dynamic propagation of client-requested changes throughout the workflow while avoiding redundant computations.

Workflow Execution Model

- The workflow operates sequentially:
  1 → 2 → 3 → 4 → 5 → 6 → 7
  with internal sub-branches and controlled detours.

- Each node (step) is autonomous and self-validating: it checks only the variables it depends on.

- All communications with the client occur in natural language, generated by the LLM and reviewed by the HIL.

- Triggers and DB updates occur automatically after each state-changing action.

Standard Step Structure (used everywhere)

Every main step in this document follows the same internal organization.
Each sub-step (1a, 1b, 1c …) adheres to these blocks:

1. **Entry Guard** – When the step should be entered.

2. **Prerequisites / Checks (stop at first unmet)** – Variables or conditions that must be valid; if not, detour to the step that provides them.

3. **Actions** – Specific operations executed by actors (LLM, HIL, Client, Trigger, DB).

4. **Exit / Return Rules** – Where control proceeds next, including re-entry conditions and detour-return logic.

Change-Propagation Logic

Change	Steps Re-evaluated	Explanation
Date change	Step 2 (Date Confirmation) → (if same room free) return to caller; else Step 3	Step 2 manages all date validation; Step 3 runs only if a new room is needed.
Room	Step 3 → caller	Re-runs capacity and availability checks.
Requirements change	Step 3 → caller	Requirements hash changes trigger re-evaluation of suitable rooms.
Product / Catering	Product sub-flow within Step 4	Does not affect date or room logic.
Negotiation / Price	Step 5 only	Offer already fixed; upstream context frozen.
Deposit / Site-visit	Step 7 and sub-branches	May trigger detours back to Step 3 or 4 if the client changes date/room during

Data Integrity and Caching Rules

- **Immutable after confirmation:** Once `date_confirmed = true` and `locked_room_id` set, all previous steps become read-only until the client explicitly requests a change.
- **Hash validation:** Before each step runs, the engine compares current hashes (`requirements_hash`, `room_eval_hash`) to decide whether re-evaluation is required.
- **Thread state:** Each conversation thread carries its current workflow state, ensuring that resuming from an email or chat continuation re-enters the correct node.

Global Exit and Completion

- The workflow completes only when an event record in the DB has status = **Confirmed** and all post-confirmation actions (CRM sync, invoicing, reporting) have been executed.
- At completion, the thread state is set to **Closed**, preventing further automated triggers unless reopened manually.

Step 1 – Intake

Entry Guard
Start this step when a new inbound message is received that is *not yet associated* with an existing workflow thread (no `event_id` context or `status` field).

1a. Actor: LLM — Intent Classification

- Parse the inbound message text.
- Classify intent via the intent model.
- If intent = "Event Request" → continue.
- If any other intent (e.g. general inquiry, spam, unrelated topic) → terminate workflow creation and forward message to manual review or general inbox.
- Store preliminary classification in memory context (`intent=event_request`, `confidence_score`).

Exit Condition: Only proceed if `intent == "Event Request"` and `confidence_score ≥ 0.85`.

1b. Actor: LLM — Entity Extraction

Extract and normalize structured entities from message content:
– date or time hint (e.g. "next Friday", "12 December 2025 evening")
– number of attendees / participants
– city or venue reference (if mentioned)
– budget or price range
– specific requirements (e.g. "needs projector", "team workshop")
– contact details (name, email, phone)
– preferred language (if detected)

LLM writes a normalized JSON structure and attaches it to the context (`intake_parsed_entities`).

1c. Actor: DB — Record Creation

- Create a new record in the database collection `Events`.
- Write all parsed fields into this record under state `Intake`.
- Initialize `status="Draft"` and `thread_id=<conversation id>`.
- Persist `created_at`, `language`, `contact data`.

- Return `event_id` to workflow memory context.

1d. Actor: Condition — Date Presence Check

Determine if the message contains an explicit and unambiguous date/time value:
• If no date found or date ambiguous (e.g. "sometime in December") → detour to **Step 2 (Date Confirmation)**.
• If valid date/time found → save as `chosen_date (candidate)` and proceed to **Step 3 (Room Availability)**.

Store in DB: `intake_stage_complete=true`, `next_step_id` accordingly.

Re-entry / Detour Rules

- If during later steps (client changes requirements or contact details), updates are written back into the same intake record for data consistency.
- Intake never re-runs fully after initial creation (except manual correction by HIL).

Exit / Transition

If no date present → Step 2 (Date Confirmation).
If date present → Step 3 (Room Availability).
Thread status updates to `In Progress`.

Step 2 – Date Confirmation (final rules)

Entry Guard
Enter this step whenever a new event request has no confirmed date (`date_confirmed != true`) or when the client later requests a date change.
Inputs required: intake record with basic event info and (if applicable) the client's initial date proposal(s).

2a. Actor: LLM — Compute Available Dates

- Query calendar and business hours to calculate the next five available dates (venue-wide, not room-bound).
- Apply blackout rules, public holidays, and maintenance blocks.
- Save the result to `candidate_dates = [d1…d5]`.

2b. Actor: UI Adapter → Client — Present Options and Invite Proposal

Send a single message containing:

- The five next available dates (`candidate_dates`).
- An explicit invitation for the client to propose a different date if preferred.
- A polite prompt such as "Here are our next available dates — would one of these work, or do you have another date in mind?"

2c. Actor: Client → LLM — Provide Response

The client replies with either a single proposed date or multiple date options.
LLM captures the response text and extracts any temporal expressions.

2d. Actor: LLM — Normalize and Parse Reply

Convert parsed dates to ISO format → `D = [d1…dn]`.
Attach timezone context from the client's profile or email metadata.
Persist to temporary variable `proposed_dates`.

2e. Actor: Condition (date availability check)

For each `d ∈ D`, check basic feasibility:

- Is it within opening hours?
- Is it not on a blackout day?
- Do buffers (before/after) allow sufficient setup/cleanup time?

Decision branch:

- If none feasible → go to 2f.
- If exactly one feasible `d*` → go to 2g.
- If multiple feasible → go to 2h.

2f. Actor: LLM → UI Adapter → Client (while loop)

Explain that none of the proposed dates work and offer a new set of five available dates.
Ask for another proposal:

"Unfortunately, those dates are not possible. Here are our next available options: […] Would one of these work?"

Loop back to 2c until a feasible date is found.
This loop may repeat until `feasible == true`.

2g. Actor: LLM → UI Adapter → Client (single date case)

If the client proposed a single feasible date `d*`, send final confirmation prompt:

"Your proposed date {d*} works for us. Should we continue with that date?"

Wait for client reply:
- If client confirms → go to 2i.
- If client changes date → return to 2c.

2h. Actor: LLM → UI Adapter → Client (multiple proposals case)

If several dates are feasible, respond explicitly:

"Date 1 unfortunately doesn't work, but Date 2 works perfectly. Should we continue with Date 2?"

On client reply "yes/confirm" → go to 2i.
If client selects another date → return to 2e for feasibility check.

2i. Actor: DB — Persist Confirmed Date

Store `chosen_date = d*`, set `date_confirmed = true`.
Record timestamp and actor (`confirmed_by = client`).
Mark date source as "client confirmed final date".

2j. Flow Transition

Once a date is confirmed and persisted:
– If no room is yet selected → proceed immediately to **Step 3 (Room Availability)**.
– If a room is already locked from an earlier iteration → skip availability check and return to `caller_step` (typically Step 4 – Offer).

2k. Re-entry Rule

If the client later cancels or revises the date (before final confirmation in Step 7):
→ control returns to Step 2 to establish a new `chosen_date`.
Upon completion, workflow resumes at `caller_step`.

Exit / Transition Summary

- **Confirmed date obtained** → Step 3 (Room Availability).
- **Client still choosing** → loop within Step 2 (2f–2h).
- **New date proposal after confirmation** → re-entry via detour rule.

Step 3 – Room Availability

(Aligned with Workflow v3 PDF + Lindy instance; includes explicit return-to-Step-2 behavior for new dates and A–C entry guard.)

Entry Guard
Enter Step 3 **only if** at least one of the following holds:
A) No room chosen yet (`locked_room_id = null`).
B) The client asks to **change the room** (e.g., "different/bigger room").
C) Any **key requirement** has changed since the last room evaluation (one of: `number_of_participants`, `seating_layout`, `event_duration`/start–end, `special_requirements`).

If none of A–C is true, **do not** run Step 3. Continue the workflow at the step that initiated the detour (use `caller_step` to return exactly there, e.g., Step 4 – Offer).

Prerequisites / Checks (stop at first unmet)

3.0a. **Confirmed Date Present**

- `chosen_date` exists with `date_confirmed = true`.
- If missing → set `caller_step = 3` and **detour to Step 2 (Date Confirmation)**. When Step 2 completes, return to Step 3 per Step-2 exit rules.

3.0b. **Requirements Snapshot Available**

- Ensure current `requirements` object is present (`number_of_participants`, `seating_layout`, `event_duration` (or start/end), `special_requirements`).
- If any required field is unknown and blocks capacity evaluation, collect minimally in-chat (LLM → Client), persist to DB, then continue.

3.0c. **Evaluation Hashing**

- Compute `requirements_hash` for the current requirements.
- Compare with `room_eval_hash` (if exists) to detect whether a re-evaluation is needed.

Actions

3a. **Actor: Condition — Build Event Window**
Construct the event window using `chosen_date`, `event_duration`, and buffer values (before/after) required by operations (setup/teardown).

3b. **Actor: LLM — Availability + Capacity Analysis**

- Use the collected information sheet and the **ROOMS** database to evaluate candidate rooms against:
  • **Calendar**: room free on `chosen_date` within event window;

• **Capacity**: room meets or exceeds `number_of_participants` for the intended `seating_layout`;
• **Constraints**: `special_requirements` compatibility (e.g., projector, stage).

- For unspecified layout, apply the default policy: **assume "XXX seating"** capacity for the room.
- If multiple rooms meet all criteria, prioritize classification **"Available"** over **"Option"**.

3c. **Actor: LLM (Decision Logic) — Classify One of Three Outcomes**

- **Unavailable**:
  - Preferred room (if specified) is busy/confirmed **or**
  - No available room matches the participant count/requirements.
- **Available**:
  - Preferred room is available (or none specified) **and** at least one room matches capacity/requirements.
- **Option**:
  - Preferred room, or all candidate rooms, are currently **on option** (penciled), with capacity fit.

3d. Conditional Branches

3e. Branch: Unavailable

– **Actor: LLM** drafts a polite unavailability reply, including:
• Thanks for the request;
• Clear statement that the requested date has no suitable room available;
• Invitation to propose **alternative dates** or to adjust requirements (e.g., attendee count, layout).
– **Actor: User (HIL)** reviews/approves and sends to the client.
– **Actor: Client** may respond with **new proposed date(s)** or **revised requirements**.
– **Actor: Condition (loop controller)**
• If the client proposes **new date(s)** → **set `caller_step = 3` and jump to Step 2 (Date Confirmation)** to confirm exactly one new `chosen_date`.
• After Step 2 completes, apply Step-2's routing:
– If the same room is kept and the date is free → return to **`caller_step` (3)** only if a room search is still required; otherwise route back to the original caller via `caller_step`.
– If a room search is needed (no room chosen yet or client wants a different room) → re-enter **3a** with the new date.
• If the client **changes requirements** (participants/layout/duration/requirements) → remain in **Step 3** (entry guard **C** satisfied) and re-run 3a–3c.
• If no new date arrives within the SLA window → set thread state **Awaiting Client Response** and pause automation.

3f. Branch: Available

– **Actor: LLM** drafts an **availability confirmation** message that explicitly states:
• The requested room/date is **available**;
• Capacity and layout **fit**;
• Any key constraints are satisfied (if applicable).
– **Actor: User (HIL)** approves and sends to client.
– **Actor: Client** replies with either a **confirmation to proceed** or a **new date proposal** or a **room change request**.
– **Actor: Condition (response routing)**
• If the client **confirms** the available room/date →
– Persist `locked_room_id` and update `room_eval_hash = requirements_hash` for this date.
– **Flow transition** → proceed to **Step 4 (Offer)**.
• If the client **proposes a different date** →
– **set `caller_step = 3`** and **jump to Step 2 (Date Confirmation)** to reconfirm a new `chosen_date`; after Step 2 finishes, route per Step-2 exit rules (skip Step 3 if a room search is not required; otherwise re-enter 3a).
• If the client **asks to change room** →
– Stay in **Step 3** (entry guard **B** satisfied) and re-run availability for alternative rooms on the **current confirmed date**; if none fit, ask whether to change the date (detour to Step 2).

3g. Branch: Option

– **Actor: LLM** drafts a message stating that the preferred room (or all rooms) are **on option**, and that capacity/requirements **fit**. Offer to proceed with the option **or** adjust.
– **Actor: User (HIL)** approves and sends to client.
– **Actor: Client** replies with one of: **accept option**, **change date**, **change room/requirements**.
– **Actor: Condition (response routing)**
• If client **accepts option** →
– Persist `locked_room_id` and `room_eval_hash` and **continue to Step 4 (Offer)**.
• If client **asks for a different date** →
– **set `caller_step = 3`** and **go to Step 2 (Date Confirmation)**; after Step 2, route per Step-2 rules.
• If client **changes requirements** (participants/layout/duration/requirements) →
– Stay in **Step 3** (entry guard **C** satisfied) and re-run 3a–3c.

Exit / Return Rules

- **To Step 4 (Offer)**:
  - When client **confirms** an **Available** room/date, or **accepts** an **Option** outcome.
  - Persist: `locked_room_id`, `room_eval_hash = requirements_hash (snapshot)`, and any info the Offer depends on (participants, layout, timing).

- **To Step 2 (Date Confirmation)**:
  - When the client proposes **new date(s)** at any time in Step 3 (Unavailable/Available/Option).
  - Always set `caller_step = 3` before detouring. After Step 2 finishes, return according to Step-2 exit rules:
    • If **room remains the same** and the new date is confirmed for that room, **skip Step 3** and return to the step that initiated the detour (via `caller_step`).
    • If **room search is needed** (no room chosen, or client asked to change room), re-enter **Step 3** on the new date.
- **Awaiting Client Response**:
  - In all branches, if the client does not respond within the operational SLA, persist state and pause triggers until a reply is received.

Persistence & State Updates (DB)

- On **Available/Option** confirmation:
  - `locked_room_id = <room_id>`;
  - `room_eval_hash = current requirements_hash`;
  - `room_decision_at = <timestamp>`;
  - `room_decision_by = client`.
- On **Unavailable**:
  - `room_status = "No match"`;
  - Optional: record top N near-miss rooms with reasons (capacity shortfall, calendar conflict).
- On **Any detour to Step 2**:
  - `caller_step = 3`;
  - `date_change_requested = true`.

Messaging Guidelines (LLM → Client; HIL approval for sends)

- **Unavailable**: empathetic and constructive; always include an ask for alternates:
  "Thanks for your request. Unfortunately, no suitable room is available on {date} for {attendees} with {layout}. Would one of these alternative dates work, or would you like to adjust attendee count/layout?"

- **Available**: concise and affirmative; confirm fit and ask to proceed:
  "Good news — {Room A} is available on {date}. It comfortably fits {attendees} in {layout}. Shall we proceed with this room and date?"

- **Option**: explain clearly and propose next step:
  "{Room A} is currently on **option** for {date}. Capacity and layout fit your needs. We can proceed under this option or consider other dates/rooms — what would you prefer?"

(All outbound client messages drafted by LLM must be approved by HIL before sending.)

Step 4 – Offer (Professional Quote)

(Aligned with Workflow v3 PDF + Lindy instance; includes products mini-flow, no redundant checks, and explicit detours.)

Entry Guard
Enter Step 4 when the intent is to produce/send an offer (i.e., after Step 3 yields **Available/Option** and the client agrees to proceed, or when returning here from a detour).
Do **not** re-check earlier steps unless their inputs changed (see checks below).

4a. Actor: Condition — Prerequisites / Checks (stop at first unmet)

P1. Date confirmed: `date_confirmed = true`.
- If **missing** or the client just requested a **date change** in this thread:
  – Set `caller_step = 4` → detour to **Step 2 (Date Confirmation)**.
  – When Step 2 completes, **return to Step 4** (skip Step 3 if same room remains; otherwise Step 3 then back to Step 4 per guards).

P2. Room outcome valid for current requirements: `locked_room_id` exists and `requirements_hash` equals the `room_eval_hash` used when this room/date was validated.
- If **missing/stale** or the client **asked to change room**:
  – Set `caller_step = 4` → detour to **Step 3 (Room Availability)**; return to Step 4 after completion.

P3. Attendee number/range present (capacity dependency).
- If **missing**:
  – Set `caller_step = 4` → detour to **Step 3 (Room Availability)** to run the required-info chat loop (LLM ↔ Client) and persist; return to Step 4.

P4. Products phase completed (client either selected from the current catalog or expressed end-intent like "no / skip / continue").
- If **not completed** → go to **4b** (products mini-flow), then return to **4a**.

No-redundant-checks rule: Do **not** re-run Step 2 or Step 3 unless P1/P2/P3 fail due to an actual change or missing input.

4b. Actor: LLM → Client (same chat) — Products / Catering Mini-Flow (loop until end-intent)

Purpose: Allow the client to add zero or more products. Current catalog = **Catering JSON** (packages, beverages, add-ons). More categories can be added later without changing the flow.

Flow

4b.1 **Ask once**:
"Would you like to add any products to your event? Right now we offer **catering options**. If not, you can say 'no', 'skip', or 'continue'."

4b.2 **If client says end-intent (no / skip / continue / move on / next step / we're good) →** `selected_products = []`; **return to 4a**.

4b.3 **If client wants products → list catalog exactly as in the Catering JSON** (names/prices/descriptions unchanged):
- **catering_packages** (6 packages, price_per_person).
- **beverages** (non-alcoholic per-person; alcoholic per-glass/per-bottle with quantity prompts).
- **add_ons** (fixed price items).

For Lunch/Premium Lunch: capture **dietary_options** and **main_course_options** if applicable.

4b.4 **Selection & Quantity**
- For **per-person** items, default quantity = `number_of_participants` unless the client specifies otherwise.
- For **per-glass/bottle** items, ask for **counts** ("How many glasses/bottles?").
- Persist each pick with: `id/name`, `unit_type` (per_person / per_glass / per_bottle / fixed), `unit_price`, `quantity`, and any `notes` (dietary).

4b.5 **Summarize & Confirm (loop)**
- Compute **catering_subtotal** =
  Σ(packages.price_per_person × participants) +
  Σ(non_alc.price_per_person × participants) +
  Σ(alcoholic.quantity × unit_price) +
  Σ(add_on.price).
- Show a concise line-item summary and ask:
  "Does this look right, or would you like to add/change anything? If you're finished, say something like 'no more' or 'continue'."

4b.6 **Branch**
- If client adds/changes → update selection, **re-summarize**, and **repeat 4b.5**.
- If client gives end-intent → **persist `selected_products`**, **store `catering_subtotal`**, and **return to 4a**.

Stop Condition: Only ends when the client signals end-intent. No HIL is required inside 4b (chat remains client↔LLM).

4c. Actor: LLM — Compose Professional Offer

Content guidelines (use venue copy tone; no invented SKUs or prices beyond DB/catalog):

- **Header**: Event title / client name / contact / thread reference.
- **Date & Time**: `chosen_date`, start–end window, buffers if relevant.
- **Room**: name (`locked_room_id`), capacity note vs. `number_of_participants`, seating layout, included room facilities.
- **Technical setup**: baseline inclusions or required add-ons (if any).
- **Catering** (optional): list selected packages, beverages, add-ons with unit pricing and computed quantities; show **catering_subtotal**.
- **Financials**:
  - If your policy includes **room rate** here, display it as a line item; otherwise mark room pricing as **TBD at confirmation**.
  - Display taxes/VAT if configured.
  - Show **Offer Total** = room (if applicable) + **catering_subtotal** + configured fees/surcharges.
- **Terms**: option validity, deposit policy, cancellation terms, and next steps (e.g., "accept in reply to proceed to confirmation").
- **Call to action**: "Please reply **accept**, **negotiate**, or **decline**. You can also change date/room/attendees at any time."

Drafted offer text remains **editable** by HIL before sending.

4d. Actor: User (HIL) — Review & Approval

- Validate tone, accuracy, pricing, policy compliance.
- Adjust any commercial detail (e.g., add a manual room price if needed).
- Approve for sending or request LLM revision (loop within 4c–4d until approved).

4e. Actor: DB — Offer Record Creation

- Create `Offer` entity:
  - `status = "Lead"`
  - `event_id`, `locked_room_id`, `chosen_date`, `requirements_hash`, `room_eval_hash`
  - `selected_products` snapshot + `catering_subtotal`
  - pricing breakdown and terms text used in the client offer

- Persist offer version (`offer_version_n`) for audit; store PDF/HTML as needed.

4f. Actor: Trigger — Send Offer to Client

- Dispatch approved offer via configured channel (email or in-thread message).
- Log delivery status and timestamp; set thread state to **Awaiting Client Response**.

4g. Actor: Client — Reply Outcomes (three canonical types)

- **Accept** (explicit): "we accept", "go ahead", "book it", etc.
- **Negotiate** (adjust commercial terms): "can we lower X", "can we include Y", "discount?", etc.
- **Decline**: "not moving forward", "we'll pass", etc.
- *(Non-decision questions are handled, but do not change 4g type; see 7 for full response analysis.)*

4h. Actor: Condition — Route on Client Reply

If Accepted → Step 7 (Event Confirmation).
If Negotiate → Step 5 (Negotiation / Close).
If Declined → End workflow (status **Lost**).

If client requests a change instead of a pure 3-way outcome:

- **Date change** → set `caller_step = 4` → **Step 2 (Date Confirmation)** → on completion, **return to Step 4** (skip Step 3 if same room kept; otherwise Step 3 then back to 4).
- **Room change / Requirements change** → set `caller_step = 4` → **Step 3 (Room Availability)** → return to **Step 4** after re-evaluation.
- **Products change** (after an offer exists) → remain in **Step 4**; re-open **4b** mini-flow, then regenerate **4c** and follow **4d–4f** again.

Persistence & State Updates (DB)

- On entering Step 4 with valid prerequisites:
  - snapshot `requirements_hash` and `room_eval_hash`;
  - ensure `locked_room_id` and `chosen_date` are bound to the current offer draft.

- On sending offer (`4f`):
  - `offer_status = "Sent"`;
  - `offer_sent_at = timestamp`;
  - link `message_id` / delivery metadata.
- On client reply (`4g`):
  - update `offer_status` accordingly (`Accepted` / `Negotiation` / `Declined`);
  - if detours triggered, write `caller_step = 4`, and `change_type = <date|room|requirements|products>`.

Messaging Guidelines (LLM → Client; HIL approvals required for sends)

- **Offer send**: confident, concise, professional; list clear next steps:
  "Attached is your offer for {Room} on {Date}. It includes {brief summary}. Please reply 'accept', 'negotiate', or 'decline'. You can also adjust date/room/attendees at any time."

- **Products loop**: friendly and efficient; never overwhelm with all details at once; show compact lists and summary totals; always honor end-intent to proceed.
- **Change handling**: acknowledge and confirm routing:
  "Happy to switch the date. I'll check availability for the same room and get right back to you." *(Detour to Step 2 with `caller_step=4`.)*

Exit / Return Summary

- **Primary forward path**: Step 4 → (Client Accepts) → **Step 7**.
- **Negotiation path**: Step 4 → (Negotiate) → **Step 5**.
- **Decline path**: Step 4 → (Declined) → **End (Lost)**.
- **Detours** (change requests):
  - Date → **Step 2** → return to **Step 4** (or via 3 if room changed).
  - Room/Requirements → **Step 3** → return to **Step 4**.
  - Products → remain in **Step 4** (4b→4c→4d→4f loop).

Step 5 – Negotiation / Close

(Aligned with Workflow v3 PDF + Lindy logic; manages counteroffers, revisions, and terminal acceptance/decline routing.)

Entry Guard
Enter Step 5 when the client's response to the offer (Step 4) expresses a desire to negotiate or alter commercial terms **without abandoning** the event request.
Triggers:

- Client explicitly says "can you adjust the price," "can we change catering," "could we get a discount," "maybe smaller package," etc.
- LLM classifier marks intent = `Negotiation`.
  If the client instead changes a structural parameter (date, room, requirements) → handle via Step 2 / Step 3 detour before returning to 4 or 5.

5a. Actor: LLM — Interpret Client's Counterpoints

- Parse the reply and classify negotiation type:
  - Price adjustment (request discount or alternative cheaper package).
  - Scope change (fewer attendees, remove items).
  - Value increase (upgrade package or add service).
  - Condition change ("can we pay deposit later", "can we move option deadline").
- Extract proposed changes to quantifiable values (price, products, participant count, etc.).
- Store to temporary variable `negotiation_proposal`.

5b. Actor: LLM → User (HIL) — Draft Counteroffer

- Generate a **bounded counteroffer**, adhering to venue pricing policy and margin constraints (from the DB or configuration).
- Include rationale (e.g., "discount of 5% for events over 50 guests").
- Suggest alternative packages if price target < minimum threshold.
- Add one clear acceptance path: "Please reply 'accept' to confirm, or 'no thanks' to decline."
- HIL reviews, edits, and approves message.
  - If HIL approves → send counteroffer to client.
  - If HIL rejects or needs change → loop back to 5b (draft revision).

5c. Actor: Client — Respond to Counteroffer

Client replies with one of:
• **Accepts** (new terms).
• **Declines** (closes negotiation).
• **Proposes another counter** (restart loop).

5d. Actor: Condition — Routing Logic

If Accepted:

- Persist final negotiated values to Offer record: `final_total`, `discounts`, `adjusted_products`, `notes`.
- Mark `offer_status = Accepted`.
- **Proceed to Step 7 (Event Confirmation).**

If Declined:

- Mark `offer_status = Lost (Negotiation Declined)`.
- Terminate workflow.

If Client Proposes Another Counter:

- Loop back to 5a (LLM parses new terms).

5e. Detour / Dependency Handling

During negotiation, if client changes structural inputs:
• **Date change** → set `caller_step = 5` → Step 2 (Date Confirmation) → return to Step 4 (Offer rebuild) → resume 5 if necessary.
• **Room change or requirements change** → set `caller_step = 5` → Step 3 (Room Availability) → then Step 4 (Offer) → 5.
• **Product changes** → remain within Step 4 subflow (4b→4c→4f) then resume negotiation context.

5f. DB Persistence & Audit

- For each negotiation round (5a→5c):
  • Append to `negotiation_history` array [{round_id, client_text, llm_draft, hil_revision, sent_at, client_response, delta}].
  • Increment `negotiation_round_count`.
- On acceptance: store `final_pricing_hash` and `final_offer_text`.

- On decline: `closed_reason = client_declined_after_negotiation`.

5g. Messaging Guidelines (LLM ↔ Client)

- Maintain positive, professional tone; never argue.
- Offer concrete alternatives rather than open questions.
- Sample LLM phrasing:
  > "I can adjust the catering package to the Basic Coffee & Tea option for CHF 8.50 per person. That brings your total to CHF … Would you like to proceed with this offer?"
- Ensure HIL approves before sending.

Exit / Return Summary

- **Accepted → Step 7 (Event Confirmation).**
- **Declined → End (status Lost).**
- **Counter continues → loop within Step 5.**
- **Detours (date/room/requirements/product changes)** → handle via Steps 2–4 then return here using `caller_step = 5`.

Step 6 – Transition Preparation

(Aligned with Workflow v3 PDF + Lindy logic; prepares transition from commercial offer phase to event confirmation. Ensures all dependencies, state sync, and context continuity before Step 7.)

Entry Guard
Enter Step 6 automatically after:

- The negotiation loop (Step 5) has **ended with acceptance**; or
- An offer (Step 4) was **accepted without negotiation**; or
- A detour from Step 7 requested a context refresh before confirmation (e.g., after deposit handling, site visit, or date change).

Step 6 acts as a **technical synchronization checkpoint**, not a client-facing step. No user messaging occurs here unless data inconsistencies are found.

6a. Actor: DB / System — State Synchronization

Perform a complete synchronization between workflow context and database records:

Check & align:
- `chosen_date` (confirmed, not stale).
- `locked_room_id` (confirmed, valid against room_eval_hash).
- `requirements_hash` (matches last evaluation).
- `selected_products` snapshot identical to accepted offer content.
- `final_total` and pricing fields.
- Thread metadata (`thread_id`, `offer_id`, `client_id`, `status`).

If any mismatch or missing field detected:
– Correct automatically if possible (re-pull from Offer record).
– If correction ambiguous → flag to HIL for manual verification (e.g., "room mismatch, please verify").

6b. Actor: Trigger — Data Preparation and Locking

After synchronization, Trigger executes:
- Create or update temporary `Event Confirmation Payload`:
  `{event_id, client_info, chosen_date, locked_room_id, total_price, selected_products, deposit_policy, site_visit_required}`.
- Lock thread against duplicate transitions (prevent re-trigger from Step 4 or Step 5 while Step 7 is active).
- Set workflow marker: `transition_ready = true`.

6c. Actor: DB — Logging and Versioning

- Create new log entry in `workflow_audit`:
  `{event_id, step=6, action="transition", timestamp, summary, previous_step}`.
- Copy essential Offer fields into the main Event record:
  `offer_version_final`, `offer_total_final`, `accepted_at`, `accepted_by`.
- Update Event status to `Transition Pending`.

6d. Flow Transition

Once all checks and synchronizations succeed:
– System sets `next_step = 7 (Event Confirmation)`;
– Workflow context variable `caller_step` cleared (no pending detour);
– Trigger dispatches message to continue workflow execution.

If synchronization fails due to missing data and cannot auto-correct, Step 6 pauses and flags the issue to HIL:
> "Transition halted: missing locked_room_id or inconsistent offer data. Please resolve before continuing."

6e. Messaging Rules (Internal only)

- No direct client communication.
- Notifications limited to system logs and HIL dashboard alerts.
- Must never duplicate offer or trigger double-confirmation.

Exit / Return Summary

- **All data validated and locked** → proceed to **Step 7 (Event Confirmation)**.
- **Data inconsistency detected** → flag to HIL, pause automation.
- **No detours or loops**; Step 6 runs exactly once per accepted offer before Step 7.

Step 7 – Event Confirmation

(Fully expanded with all branches: client responses, site visit, reservation, deposit, and final confirmation — aligned with Workflow v3 PDF + Lindy instance.)

Entry Guard
Enter Step 7 whenever the **client has replied to an offer or negotiation** with an intent indicating one of the following:

- "We'd like to confirm / book / proceed."
- "We'd like to visit the room before confirming."
- "Please reserve the date for now."
- "Can we change something?" (date, room, attendees, catering, technical, etc.)
- "We're not interested anymore."
- "Just one more question…"

Step 7 acts as the **unified post-offer analysis node**, classifying the client's message and routing to the correct handling branch.

7a. Actor: LLM — Response Analysis

LLM parses the incoming client reply and classifies it into one of six standard **Response Types**:

Type	Description
1. Confirm Booking	Client explicitly accepts and confirms the offer.
2. Site Visit Requested	Client requests to view the venue before booking.
3. Reserve Date	Client wants a provisional hold on the date (option).
4. Change Request	Client wishes to alter details (date, room, attendees, products, etc.).
5. Cancel / Decline	Client withdraws or states no further interest.
6. Question / Clarification	Client asks a question without making a decision.

If no clear intent is detected → LLM sets `status = unclear` and routes to HIL for manual review.

7b. Conditional Branch Routing

Each Response Type maps to a dedicated branch (7.1–7.6).
At any moment, if a change affects earlier workflow variables, Step 7 can trigger **detours to Steps 2–4**, then resume at Step 7 after dependency resolution (caller_step = 7).

Branch 1 – Client Confirms Booking

Actors & Actions

1. **LLM → Trigger / DB**
   - Verify that the accepted offer exists, is current, and matches all variables (`chosen_date`, `locked_room_id`, `requirements_hash`).
   - Check if a deposit is required (policy flag).
   - Create initial calendar event (status = *Option*).
   - If **no deposit required** → immediately mark event **Confirmed**.

2. **LLM → User (HIL)**
   - Draft final confirmation email summarising the agreed date, room, and services.
   - HIL reviews and approves.

3. **Trigger → DB**
   - Update event record:
     `status = Option` or `Confirmed` (depending on deposit policy).
     `confirmation_source = client_acceptance`.
     `confirmation_timestamp = now()`.

Exit / Return

- If **deposit required** → go to Branch 3 (Deposit / Reservation).

- If **confirmed without deposit** → skip directly to Finalisation (7j–7k).
- If **client later changes date or room after confirming** → set `caller_step = 7`, detour to Step 2 or 3, then re-enter Step 7.

Branch 2 – Site Visit (Viewing) Requested

Preconditions

- Venue supports site visits (`site_visit_available = true`).
- Offer and chosen_date are consistent.

Actions

1. **LLM → Trigger**
   • Retrieve possible visit times around `chosen_date` or next open slots.
2. **Trigger → LLM → HIL**
   • LLM drafts a message:
   > "We'd be happy to arrange a site visit. Here are some possible times: [options]. Which would suit you?"
   • HIL approves and sends to client.
3. **Client → LLM**
   • Client chooses or proposes an appointment.
4. **Trigger → DB**
   • Schedule site visit in calendar.
   • Create visit entry (`visit_id`, `scheduled_at`, `status = Scheduled`).
5. **After the visit:**
   • If client **confirms booking** → go to Branch 1 (Client Confirms).
   • If client **requests changes** → detour to Steps 2–4 based on change type, then return to Step 7.
   • If client **declines after visit** → go to Branch 5 (Cancel / Decline).

Exit / Return

- Once the site visit concludes, system re-enters Step 7 for reevaluation of client intent.

Branch 3 – Reserve Date (Provisional Hold / Deposit Handling)

Purpose
Allow the client to reserve the date temporarily while finalising payment or decision.

Actions

1. **LLM → Trigger → DB**
   - Create calendar event with `status = Option`.
   - Assign `option_valid_until = now() + policy_days`.

2. **LLM → HIL → Client**
   - Draft message confirming provisional reservation and next steps:
     > "We've reserved {Room} on {Date} for you. The option is valid until {ExpiryDate}. Once we receive your deposit, your booking will be confirmed."
   - HIL approves and sends.

3. **Client → Trigger**
   - If client pays deposit before expiry → Trigger updates event: `status = Confirmed`.
   - If deposit not received by expiry → Trigger auto-cancels reservation (status = Lost).

4. **DB updates**
   - On deposit received: `deposit_received_at`, `payment_reference`.
   - On expiration: `option_expired_at`.

Exit / Return

- Deposit received → proceed to **Final Confirmation (7j–7k)**.
- Deposit pending → remain in Option (waiting).
- Deposit expired / declined → workflow ends (Lost).

Branch 4 – Change Request

Triggers
Client requests to change date, room, attendee count, layout, catering, or other event parameters.

Actions & Routing

1. **LLM → Identify Change Type:**
   - Date → Step 2 (Date Confirmation).
   - Room / Requirements → Step 3 (Room Availability).
   - Products / Catering → Step 4 (Offer).

2. **Set `caller_step = 7`** before detour.

3. After the resolving step completes, **return to Step 7** and re-evaluate response type:
   - If the result is still "Confirm Booking" → Branch 1.
   - If now "Reserve Date" → Branch 3.
   - If new offer created → send to client and await new Step 7 entry.

Example Dialogue
Client: "Could we move this to Friday instead?"

→ Step 7 detects date change → Step 2 executes → on new date confirmation, returns to Step 7 and re-sends availability confirmation or updated offer.

Branch 5 – Cancel / Decline

Actions

- **LLM → HIL → Client**
  Draft polite closure:
  > "Thank you for considering us. We've released the date, but we'd be happy to help with future events."

- **DB**
  Update event record:
  `status = Lost`,
  `lost_reason = client_declined / cancellation`,
  `closed_at = now()`.

- **Trigger**
  Remove any temporary calendar holds (status = Cancelled).

Exit / Return
→ End of workflow.

Branch 6 – Question / Clarification

Purpose
Handle neutral inquiries where the client has not yet decided.

Actions

- **LLM → HIL → Client**
  Provide clear, factual answer to the question (e.g., "Does the room have a projector?").

- **Thread remains active**, `status = Awaiting Client Response`.

- When client replies again, Step 7 re-runs classification and routes to the appropriate branch.

7j. Post-Confirmation / Deposit Handling (Aggregated)

This sub-phase merges all branches where a deposit may still be pending or a final confirmation must be recorded.

Actions

1. **LLM → HIL → Client**
   If deposit required but unpaid:

> "To finalise your booking, please proceed with the deposit of CHF {amount}. Once received, we'll confirm your event officially."

2. **Trigger → DB**
   Monitor for incoming payment.
   When deposit received:
   - Update `status = Confirmed`;
   - Record `payment_reference`;
   - Notify HIL;
   - Automatically trigger final confirmation message (7k).

3. **If deposit overdue** → Trigger sends reminder or cancels per policy.

7k. Finalisation / Closure

Actions

1. **LLM → HIL → Client**
   Draft and send final confirmation email:
   > "Your event on {Date} at {Venue/Room} is now fully confirmed. We look forward to welcoming you."
   Include summary of date, room, catering, total, and contact details.

2. **Trigger / DB**
   - Update Event record: `status = Confirmed`, `thread_status = Closed`.
   - Create final calendar block.
   - Sync to CRM, invoicing, reporting modules.
   - Generate event confirmation document (PDF).

Exit / Completion
Workflow terminates with:
`event_status = Confirmed`, `thread_status = Closed`, all post-confirmation actions complete.

Detour / Re-Entry Rules (apply to all branches)

- At any point, if client changes any dependency (date, room, requirements, products):
  → set `caller_step = 7` and detour to the appropriate step (2–4).
  → when resolved, re-enter Step 7 and resume classification.

- Only dependent steps re-run; all unrelated data remains cached.

- Step 7 repeats until a **terminal state** (`Confirmed` or `Lost`) is reached.

Global Detour Rules & Final Sequence Summary

(Part 10 – Completing the Workflow v3 Specification)

This final section defines the global runtime logic that governs **how steps interact**, **how dependencies propagate**, and **how the workflow reaches completion**.
It ensures that Codex or any workflow engine can maintain consistent execution without requiring diagrammatic context.

A. Global Detour Rules

Detours are temporary backward jumps to earlier workflow steps when client actions or data inconsistencies require re-validation of dependencies.
They are always **stateful**, meaning the workflow remembers its origin (`caller_step`) and returns there upon resolution.

A1. Detour Trigger Conditions

Condition Type	Trigger	Target	Explanation
Client changes **date**	3, 4, 5, 7	**Step 2**	All date-related logic is centralised in Step 2 (Date Confirmation). Step 3 and beyond depend on this variable,
Client changes **room**	4, 5, 7	**Step 3**	Step 3 owns all room and capacity checks. Other steps must detour there when the room changes.
Client changes **requirements** (attendees,	3, 4, 5, 7	**Step 3**	These parameters determine room fit and must be re-evaluated in Step 3.
Client changes **products** (catering, add-ons,	4, 5, 7	**Step 4**	Product selection lives inside the Offer logic. No need to repeat earlier steps.
Offer or negotiation	5, 6,	**Ste**	When terms or totals differ from accepted data, rebuild the
Payment / deposit change	7	**Step 7j**	Handled within Event Confirmation's deposit branch (never external detour).

A2. Detour Execution Rules

1. **Set `caller_step`** before initiating detour.
2. **Jump** to the target step to resolve the dependency.
3. On successful completion of the target step:
   - If the previous dependency (date, room, etc.) has been re-established,
   - **Return** to the original caller step using `caller_step` memory.
4. **Skip redundant evaluations**:
   If the revalidated output matches existing hashes (`requirements_hash`, `room_eval_hash`), resume directly without re-running intermediate logic.

A3. Return Behavior

From (detour)	Returns to	Condition

Step 2 (Date Confirmation)	previous `caller_step` (usually 3 or 4)	Once a new date is confirmed.
Step 3 (Room Availability)	previous `caller_step` (usually 4 or 7)	Once a valid room outcome (Available/ Option) is confirmed.
Step 4 (Offer)	previous `caller_step` (usually 5 or 7)	After new offer sent/accepted.
Step 5 (Negotiation)	Step 7	When negotiation accepted.
Step 7 (Event Confirmation)	N/A	Terminal step – no outward return.

B. Hash Validation Logic (Consistency Layer)

To avoid unnecessary re-execution, every dependency check uses **hash comparison**:

Variable	Description	Usage
`requirements_hash`	Hash of all requirement fields.	Recomputed whenever participants/layout/ duration change. If changed, Step 3 reruns.
`room_eval_hash`	Snapshot of requirements used during last room availability	Compared with current `requirements_hash`; mismatch →
`offer_hash`	Snapshot of accepted offer values.	Used during Step 6–7 sync to detect inconsistencies before confirmation.

C. Thread and Persistence Rules

Each event conversation thread carries all metadata required to resume exactly where it stopped.

Stored per thread:

- `event_id`, `current_step`, `caller_step`, `status`, `intent`, `context`,
- `requirements_hash`, `room_eval_hash`, `offer_hash`,
- `locked_room_id`, `chosen_date`, `selected_products`,
- `last_client_message`, `last_llm_message`, `pending_action`.

On resume:

– System rehydrates full state;
– Validates hashes and timestamps;
– Re-enters the correct step (`current_step`);
– Executes pending action if timeout or response awaited.

D. Global Messaging Protocols

All LLM-to-client outputs follow the same message-approval chain:
LLM → HIL (approval) → Client.

Only exceptions:
– Product sub-loop (Step 4b);
– Automatic deposit reminders (Step 7j).

Every sent message includes a hidden workflow tag (`step_id`, `intent`, `event_id`) for traceability.

E. Workflow Termination Rules

The workflow reaches its terminal state when any of the following is true:

Terminatio	Condition	Resulting Status
Confirmed Booking	Deposit received or deposit not required; final confirmation sent.	`event_status = Confirmed`, `thread_status = Closed`.
Client	Client explicitly declines or option	`event_status = Lost`,
System	No response for > policy window	`event_status = Lost`,
Manual	HIL marks thread as closed (e.g.,	`event_status = Closed_Manual`.

Once closed, the workflow instance is immutable except for read-only reporting.

F. Audit and Traceability

For every transition, append an entry in `workflow_audit`:

```
{
  event_id,
  from_step,
  to_step,
  trigger_actor,
  reason,
  timestamp,
  metadata
}
```

This allows full reconstruction of any event's lifecycle from Intake to Confirmation.

G. Final Sequence Summary (A → Z)

Normal forward flow (without detours):

```
1  Intake
2  Date Confirmation
3  Room Availability
4  Offer (Professional Quote)
5  Negotiation / Close  (optional)
6  Transition Preparation
```

```
7  Event Confirmation
    ├─ Site Visit (optional)
    ├─ Reserve Date / Deposit (optional)
    └─ Final Confirmation → End
```

With dynamic detours:

```
2 ↔ 3  Date ↔ Room adjustments
3 ↔ 4  Room ↔ Offer adjustments
4 ↔ 5  Offer ↔ Negotiation cycle
5 → 7  Acceptance → Confirmation
7 ↔ 2/3/4  Change requests before final confirmation
```

At any point, the workflow may loop through these intermediate routes but always terminates in one of two end states:

✅ **Confirmed (success)**

❌ **Lost (declined, timeout, or cancelled)**

H. System Summary for Codex Implementation

- Every main node (1–7) is a *deterministic state function* that:
  - validates its prerequisites,
  - executes actions,
  - determines next step or detour.
- All changes are transactional; a failed sub-step must roll back partial updates.
- External interfaces (Mail, Calendar, Payment, CRM) are handled via **Trigger adapters**; the LLM never manipulates external APIs directly.
- The HIL interface exposes approval, override, and manual-intervention hooks for every client-facing message or critical update.
- Codex should maintain in-memory context keys for:
  `current_step`, `caller_step`, `last_client_action`, `requirements_hash`, `room_eval_hash`, `offer_hash`, `status`, `next_step`.
- On each new message:
  1. Detect intent;
  2. Locate current step;
  3. Apply entry guards and hash checks;
  4. Execute or detour accordingly.

Prompt Appendix (LLM Draft Templates; HIL approves before send)

Conventions:
• Variables in `{curly_braces}` (e.g., `{Date}`, `{RoomName}`).
• Keep venue's voice: concise, professional, friendly.
• Never invent prices/SKUs not in DB.
• Always include a clear next step / question at the end.

1) Intake / Acknowledgement (optional)

When intent=Event Request detected and you want to acknowledge receipt.

```
Subject: Thanks for your event request

Hello {ClientName},

Thank you for reaching out about your event. I'll gather the
essentials (date, attendees, room fit) and come back to you
shortly.
If you already have a preferred date or attendee count, feel
free to share it now.

Best,
{AgentSignature}
```

2) Date Confirmation

2A. Present 5 dates + invite proposal (Step 2b)

```
Hello {ClientName},

Here are our next available dates:
• {Date1}
• {Date2}
• {Date3}
• {Date4}
• {Date5}

Would one of these work, or do you prefer a different date?
If you have alternatives, please share them and I'll check
feasibility.
```

2B. Single feasible date – ask to confirm (Step 2g)

```
Good news — your proposed date {Date} works. Should we
continue with {Date}?
```

2C. Multiple proposals with one feasible (Step 2h)

```
Date {BadDateLabel} unfortunately doesn't work, but
{GoodDateLabel} works perfectly.
Should we continue with {GoodDate}?
```

2D. None feasible – loop (Step 2f)

```
Thanks for the options. Unfortunately none of those dates are
possible.
Here are the next available dates:
• {Date1} • {Date2} • {Date3} • {Date4} • {Date5}

Would any of these work, or would you like to suggest others?
```

3) Room Availability

3A. Unavailable (Step 3e)

```
Thank you, {ClientName}. On {Date}, we don't have a suitable
room that meets {Attendees} in {Layout}.
Would you like to consider alternative dates, or adjust the
attendee count/layout? I can also suggest close alternatives.
```

3B. Available (Step 3f)

```
Great news — {RoomName} is available on {Date}. It
comfortably fits {Attendees} in {Layout}, and meets your
requirements ({RequirementsShort}).
Shall we proceed with this room and date?
```

3C. Option (Step 3g)

```
{RoomName} is currently on option for {Date}. Capacity and
layout fit your needs.
Would you like to proceed under this option, look at other
dates, or consider a different room?
```

4) Products / Catering Mini-Flow (Step 4b)

4A. Initial ask

Would you like to add any products to your event? Right now we offer catering options.
If not, you can say “no / skip / continue” and we’ll proceed.

4B. Present catalog (use exactly the Catering JSON fields/labels)

Here are our catering options (per our catalog):

Packages (per person):
- Basic Coffee & Tea — CHF 8.50 … {desc}
- Coffee & Snacks — CHF 15.00 … {desc}
- Lunch Package — CHF 28.00 … {desc}
- Premium Lunch — CHF 42.00 … {desc}
- Apéro — CHF 24.00 … {desc}
- Premium Apéro with Bar — CHF 38.00 … {desc}

Beverages:
- Non-alcoholic (per person): Soft Drinks CHF 4.50; Fresh Juices CHF 5.00; Water CHF 3.50
- Alcoholic: Wine (per glass CHF 8.00 / per bottle CHF 35.00), Prosecco (per glass CHF 9.00 / per bottle CHF 40.00), Beer (per bottle CHF 6.50)

Add-ons (fixed):
- Birthday Cake CHF 65.00
- Chocolate Fountain CHF 120.00
- Barista Service CHF 150.00

Please tell me what you’d like. For per-person items I’ll default quantities to your attendee count ({Attendees}) unless you say otherwise. For wine/beer/prosecco, please specify glasses/bottles.

4C. Summarize & confirm loop

Here’s your current selection:
{LineItemsWithQuantitiesAndUnitPrices}
Estimated catering subtotal: CHF {SubtotalEstimate}

Would you like to add or change anything, or should we continue?

4D. Dietary details (when Lunch/Premium Lunch chosen)

Do you have any dietary needs for the lunch menu (e.g., Vegetarian, Vegan, Gluten-free)? Also, for mains, please choose from:
{MainCourseOptionsList}

5) Offer Send (Step 4f) & Offer Cover Text (Step 4c)

Subject: Offer for {EventName} — {RoomName} on {Date}

Hello {ClientName},

Please find your offer below for {RoomName} on {Date}:
• Room & setup: {RoomName}, {Layout}, capacity {CapacityNote}
• Catering (if any): {ShortCateringLines}
• Inclusions / technical: {TechShort}
• Commercials: {RoomPriceIfShown} + catering subtotal = CHF {CateringSubtotal} {TaxNote}
• Terms: {OptionValidity}, {DepositPolicy}, {CancellationPolicy}

Please reply “accept”, “negotiate”, or “decline”. You can also change date/room/attendees anytime, and I’ll adjust the offer accordingly.

Best,
{AgentSignature}

6) Negotiation (Step 5)

6A. Bounded counteroffer

Thanks for your note. I can propose the following:
• {ChangeSummary} (e.g., switch to Coffee & Snacks package, −CHF X total)
New total would be CHF {NewTotal}.

If this works, reply “accept” and I’ll proceed to confirmation. Otherwise, tell me what you’d like adjusted.

6B. Alternative if below threshold

I can’t reach that exact total due to minimum pricing, but here’s the closest alternative:
• {AlternativePackageOrScope} → CHF {NewTotal}
Would this be acceptable?

7) Event Confirmation (Step 7)

7A. Client confirms booking (Branch 1)

Wonderful — we’re ready to proceed with your booking for {RoomName} on {Date}.
{IfDepositRequired: To finalise, a deposit of CHF {DepositAmount} is required. I’ll send details now.}
I’ll place the booking and send a confirmation message shortly.

7B. Site Visit request (Branch 2)

Happy to arrange a site visit. Here are possible times:
• {Slot1}
• {Slot2}
• {Slot3}

Which would suit you? If you have other preferences, let me know and I’ll try to accommodate.

7C. Reserve date / Option (Branch 3)

We’ve reserved {RoomName} on {Date} under an option valid until {ExpiryDate}.
To confirm the booking, please proceed with the deposit of CHF {DepositAmount}. I’ll send payment details now.

7D. Change request detected (Branch 4) — acknowledge & route

Got it — you’d like to change {ChangeType}. I’ll handle that now and come back with an updated availability/offer shortly.

7E. Client no longer interested (Branch 5)

Thank you for letting us know. We've released the date, and we'd be happy to assist with any future events.

7F. Question / Clarification (Branch 6)

Thanks for your question — here's the info:
{AnswerContent}

If everything looks good, feel free to "accept", "negotiate", or ask for changes (date/room/attendees).

8) Deposit Handling (7j)

8A. Deposit request

To finalise your booking, please proceed with the deposit of CHF {DepositAmount}.
Payment details: {PaymentInstructions}. Once received, I'll confirm your event officially.

8B. Deposit reminder (before expiry)

Friendly reminder: the deposit for {RoomName} on {Date} is pending.
Please complete payment by {ExpiryDate} to maintain your reservation.

8C. Deposit received → confirmation

Thank you — we've received your deposit. Your event on {Date} at {RoomName} is now confirmed.
I'll send a final confirmation email with all details next.

9) Final Confirmation (7k)

Subject: Booking Confirmed — {EventName} at {RoomName} on {Date}

Hello {ClientName},

Your event is now fully confirmed:
- Date & time: {Date}, {Start}–{End}
- Room: {RoomName}, {Layout}, capacity {CapacityNote}

```
• Catering: {ShortCateringLinesOr“None”}
• Commercials: CHF {FinalTotal} {TaxNote}
• Contact on the day: {VenueContact}

We look forward to hosting you. If anything changes, just
reply here.
Best regards,
{AgentSignature}
```

10) Awaiting Response / Nudge (generic)

```
Just checking in — did you have a chance to review the last
message?
If you’d like to proceed, reply “accept”. If you prefer
changes, I can adjust date/room/attendees or catering.
```

11) Internal (not sent to client) — Step 7 classification rubric (for Codex)

- **Confirm Booking**: phrases like “accept”, “confirm”, “book it”, “go ahead”.
- **Site Visit**: “visit”, “viewing”, “see the room”, “tour”.
- **Reserve Date**: “hold”, “reserve”, “option”, “pencil in”.
- **Change Request**: “different date”, “other room”, “increase to X people”, “change menu”.
- **Cancel/Decline**: “not moving forward”, “cancel”, “no longer interested”.
- **Question**: explicit interrogatives without a decision intent.

Coverage note

- These templates cover **every client-facing message** from Steps **2 → 7** (including products, offer, negotiation, deposit, site visit, final confirmation, nudges).
- All routing logic (detours, dependencies, hashes, caller_step) remains in the main spec you already have.
- HIL approval gates are explicitly noted where required.