AI Copilot Instructions:

Claude Code brief (from scratch)

You are building an AI Copilot that helps a user create and edit a **workflow template** via chat.

What the workflow looks like (core concept)

- A workflow template is an **ordered list of steps**.
- Steps run **top-to-bottom in order**, unless a step is a branching/control step that creates multiple paths.
- It's a **list of steps** of human actions, AI actions, system automations, and controls such as branching, which can branch into multiple paths before returning back to the main path.

Step types you must support

Human steps:

- `FORM`
- `QUESTIONNAIRE`
- `FILE_REQUEST`
- `TODO`
- `APPROVAL`
- `ACKNOWLEDGEMENT`
- `ESIGN`
- `CUSTOM_ACTION`
- `WEB_APP`

- `DECISION` (up to 3 outcomes, each with its own path)

Control steps:

- `SINGLE_CHOICE_BRANCH` (if/else)
- `MULTI_CHOICE_BRANCH` (any paths whose conditions are met are run, max 3 paths)
- `PARALLEL_BRANCH` (max 3 paths, all run in parallel)

- `GOTO` (can only be added to a path in a decision or single choice branch, points to a goto_destination)
- `GOTO_DESTINATION` (point that the goto jumps to in the flow, if jumping backward the steps are reopened)
- `TERMINATE` (end the flow, marking it as either canceled or complete)
- `WAIT` (listen for some event to be completed in another system)

Fields every step must have

All steps include:

- `id`
- `type`
- `title`
- `description`
- `assignees[]` (list of placeholders, not actual people)
- optional `due_date_relative` (example: “due in 3 days”)

- `skip_sequential_order` boolean (default false)
- `visibility` settings (keep simple for MVP)

Rules for specific step types

- **FORM / PDF Form**: can have multiple assignees; **any one** submission completes the step.
- **QUESTIONNAIRE**: exactly one lightweight question; any one submission completes; optional review gate.
- **FILE_REQUEST**: multiple files; optional reviewer; if reviewer enabled, completion requires review.
- **ESIGN**: multiple signers; can be sequential or parallel; **all required**; document becomes immutable once started.
- **APPROVAL**: cannot be declined; supports approval requirement = `ONE` / `MAJORITY` / `ALL`; can be sequential or parallel.
- **ACKNOWLEDGEMENT**: optionally requires attachment review before completion.
- **DECISION**: exactly **one** assignee; has up to **3 outcomes**, and each outcome contains its own list of steps.

Assignees (placeholders + resolution)

Assignees are not "users"—they are **roles/placeholders** that eventually resolve to an **email string**.

Default assignee: `CONTACT_TBD`.

Supported ways to resolve an assignee:

- `FIXED_CONTACT`
- `WORKSPACE_INITIALIZER`
- `KICKOFF_FORM_FIELD`

- `FLOW_VARIABLE`
- `RULES_BASED`
- `ROUND_ROBIN`

Assignee toggles:

- `allow_view_other_assignees_actions` (boolean)
- `is_coordinator` (boolean)

Also support workflow-level coordinators (not tied to a step).

Milestones (UI grouping)

- A workflow can group steps into **milestones** (just UI grouping).
- Every step belongs to a milestone.
- For MVP, you can default to **one milestone** unless user explicitly wants phases.

Hard constraints (must enforce)

- Maximum branch nesting depth: **2**
- A branch and all its paths must stay within **one milestone** (cannot span milestones).
- You cannot put milestones *inside* branch paths.
- `PARALLEL_BRANCH` has max **3** paths.
- `DECISION` has max **3** outcomes.

Goto rules:

- `GOTO` is allowed only inside:

    - a `DECISION` outcome, or
    - a `SINGLE_CHOICE_BRANCH` path
- `GOTO` can only jump to a `GOTO_DESTINATION` step that exists on the **main top-to-bottom list** (not inside a branch).

Terminate rules:

- `TERMINATE` only allowed inside a `DECISION` outcome or `SINGLE_CHOICE_BRANCH` path.
- Terminate status must be `COMPLETED` or `CANCELLED`.

Subflows are not supported

- If user asks for subflows, the Copilot must refuse and suggest flattening the process into one workflow using milestones/branches/decisions.

Kickoff and variables

Workflow start modes:

- `MANUAL_EXECUTE` (default)
- `KICKOFF_FORM`
- `START_LINK`
- `WEBHOOK`
- `SCHEDULE`
- `INTEGRATION`

Kickoff form data is globally referenceable.

Workflow variables:

- Only `TEXT` or `FILE`
- Set only at initiation time
- Immutable

AI + system automations (as steps)

Support an automation step type:

- `AI_AUTOMATION` (auto-completes; coordinator-only)
    - has inputs, prompt, optional knowledge references, and produces outputs that later steps can reference

Also define placeholders for system automation steps (you don't need to implement external calls in MVP):

- webhook, email, chat message, update workspace info, business rule mapping

How the Copilot should behave in chat

It should feel like the user building the flow has an expert AI SE available to talk to (rather than connecting with a human). The AI SE should be friendly and conversational to provide consultation to the flow builder, replacing the need for them to talk to a human (e.g. start the conversation with Hi! I am your AI SE, here to help you build and edit your flow. What would you like to accomplish today? etc).

Copilot is like an expert **solutions engineer**:

- Ask **minimal** clarifying questions only if required.
- Otherwise, apply defaults.

The Copilot must support two operations:

1. **Create** a brand new workflow from the chat prompt
2. **Edit** an existing workflow based on the chat prompt

Edits must be returned as a list of deterministic operations (patches), not as “rewrite everything”, unless the user explicitly says “rebuild from scratch”.

Output formats (strict JSON only)

The model must output exactly one of these JSON objects:

Create

{ "mode": "create", "workflow": { "...": "full workflow json here" } }

Edit

{ "mode": "edit", "operations": [{ "...": "operation 1" }, { "...": "operation 2" }] }

Clarify

{ "mode": "clarify", "questions": [{ "id": "q1", "text": "..." }] }

Reject

{ "mode": "reject", "reason": "...", "suggestion": "..." }

What to build (MVP app)

Build a small web app:

- Left side: chat interface (history + input box)
- Right side: workflow visualizer that renders:
  - the main step list in order
  - branch steps expanded to show paths/outcomes
  - milestones as section headers
- A lightweight server (or local module) that:
  - stores the current workflow JSON
  - sends user messages + current workflow JSON + constraints to the LLM
  - receives create/edit output JSON

    - validates constraints
    - applies edit operations to update the workflow
    - returns updated workflow to the UI

Notes: The workflow visualizer on the right side can start as pretty lightweight, as it is meant for display purposes (not functional) right now. In the future phase, we will make it functional. It should display branch paths nicely though, and I'll share reference examples. The main feature we want to build is the AI copilot chat to generate that workflow & edit that workflow.

Step 1:

Start with the backend.

Implement the workflow data model in TypeScript (types/interfaces), plus:

1. `validateWorkflow(workflow)` enforcing all constraints above
2. An "edit operation" format and `applyOperations(workflow, operations)` that deterministically updates the workflow
3. Unit tests for validator + applyOperations using a few sample workflows with branches/decision/goto/terminate