# System Installer

To start using the platform on your local machine, you need to go through a simple setup process using the **System Installer**. This installer runs in your browser and helps you launch, stop, or update the system via Docker.

---

# Where to Download the Installer

https://github.com/EpicStaff/EpicStaff/releases/tag/Hackathon

Download the archive for your system, unzip it and run.

MacOS only

For **macOS** currently you'll need to run commands to make application executable in unzipped directory:

**chmod -R epicstaff.app**

**xattr -r -d com.apple.quarantine**

---

# Requirements

Before launching the installer:

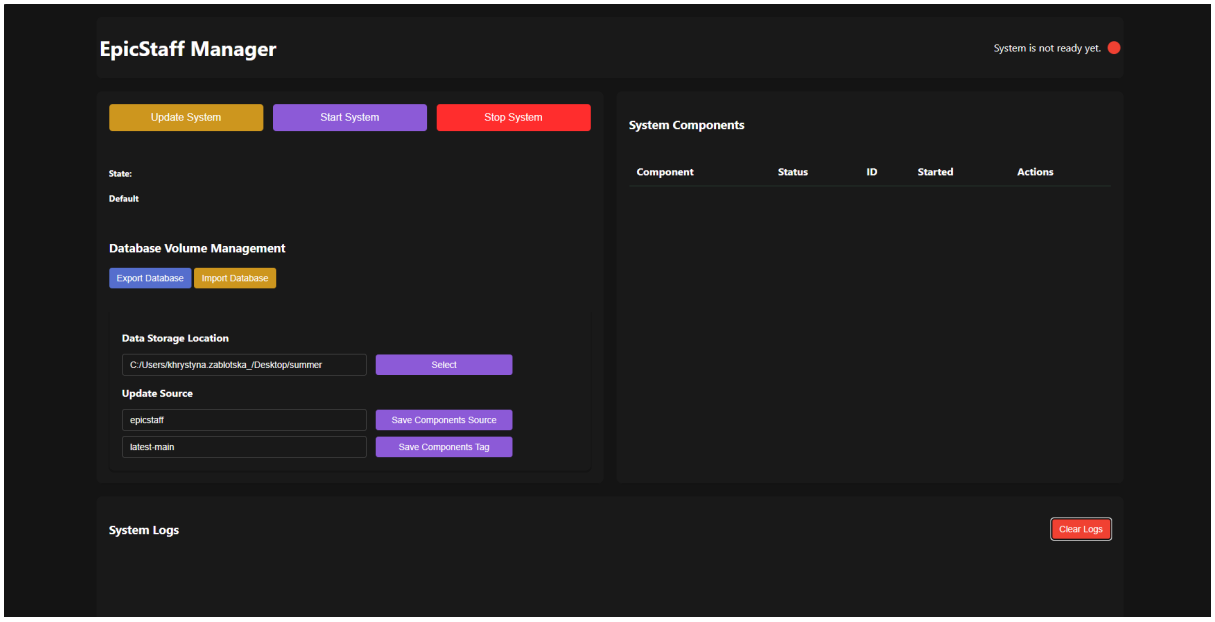
- Make sure you have **Docker** installed on your machine.
- You'll also need the **installer file** downloaded.

---

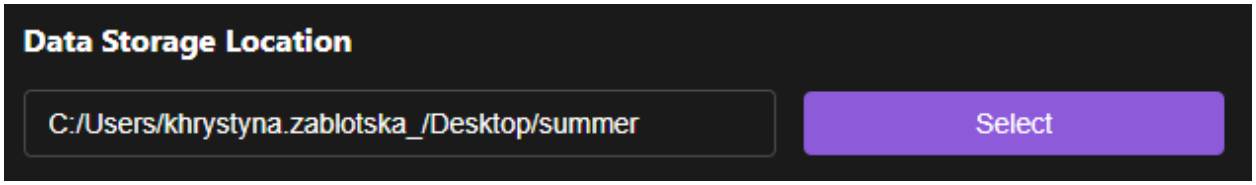
# Launching the System

1. **Open the installer file.** If Docker is not running, it will launch automatically.
2. Your browser will open the **Installer Page**.

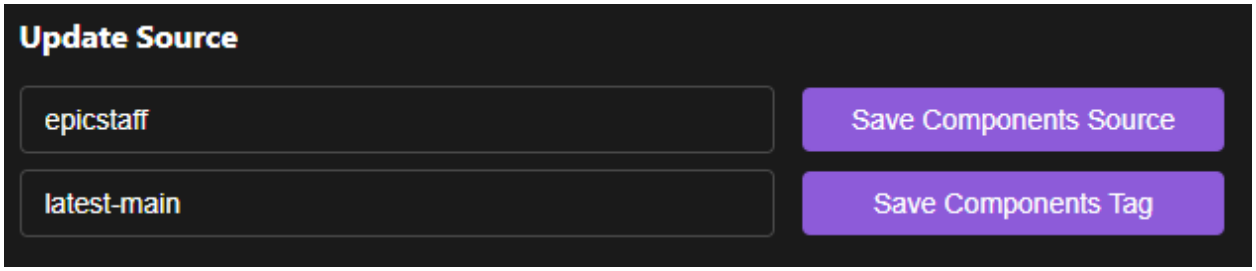
3. You'll see several main control buttons:
    - `Start System`
    - `Update`
    - `Stop System`




4. Under **Data Storage Location**, choose a folder on your computer where all system results and data will be stored by clicking "Select" button.



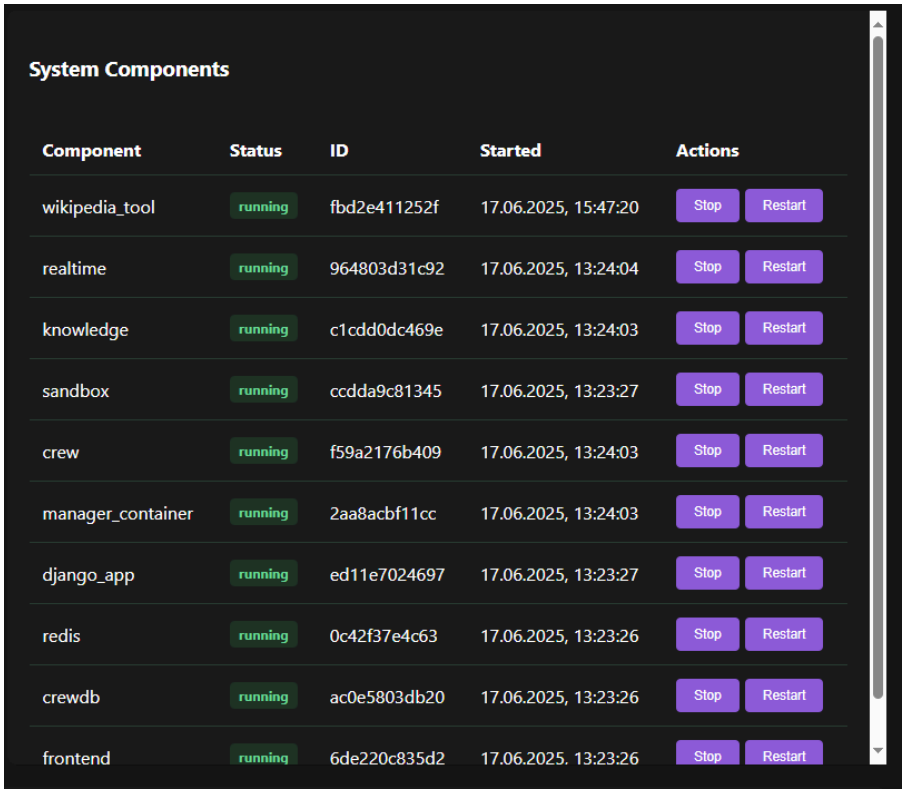

5. Under **Update Source**, fill in:
    - `Components Source: epicstaff`
    - `Components Tag: latest-main`



## Containers and Status

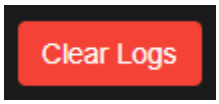
On the right-hand side, you'll see all **Docker containers** used by the system:

- Each container has a status: Running, Updating, Stopped, etc.
- You can **stop or restart** containers individually via buttons next to each one (just like in Docker Desktop).
- When the system is being updated or stopped, the containers may temporarily disappear.




At the bottom, there's a **System Logs** section:

- It shows real-time logs for everything happening during install, update, or startup.
- You can clear the logs using the **Clear Logs** button — this doesn't delete data, only clears the visible output.


---

## System States

While the system performs actions (like updating, launching, or stopping), you'll see a **System Status** message:

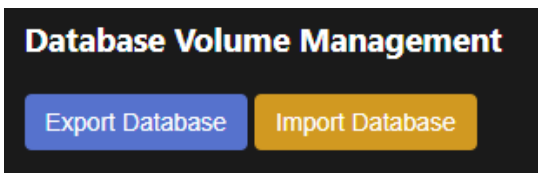
- When idle, the state is: `State: Default`
- When updating, starting, or stopping, the state will reflect that operation.


---

## Database Management

The installer also supports **Database Volume Management** with two actions:

- `Export Database` – Save your current database
- `Import Database` – Load a previous backup
  (*The import happens directly in the browser.*)



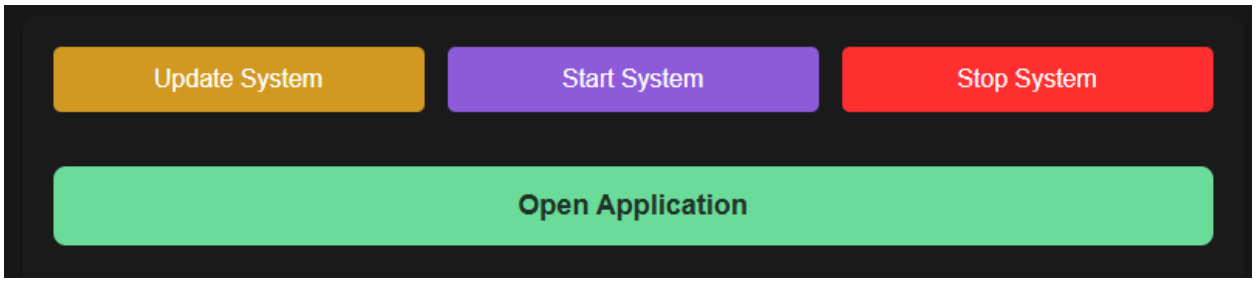

---

## Opening the Application

Once the system is started, **wait for all containers to finish loading** (you'll see them appear with `Running` status). When everything is ready, a new button will appear:

`Open Application`

Click it to enter the main application and start using the platform.




---

## ⚠️Important Notes

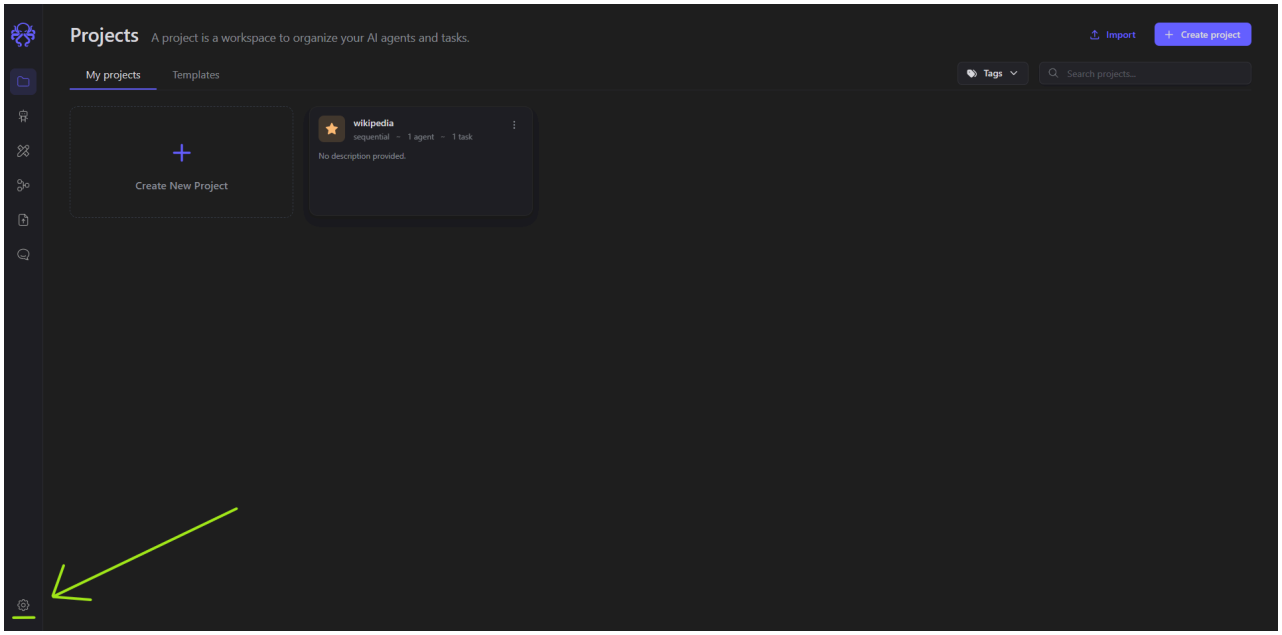
- **Updating ≠ Launching**
  After clicking `Update`, you still need to click `Start System` again to run the platform.

# LLM Settings: How to set up models (Required step)

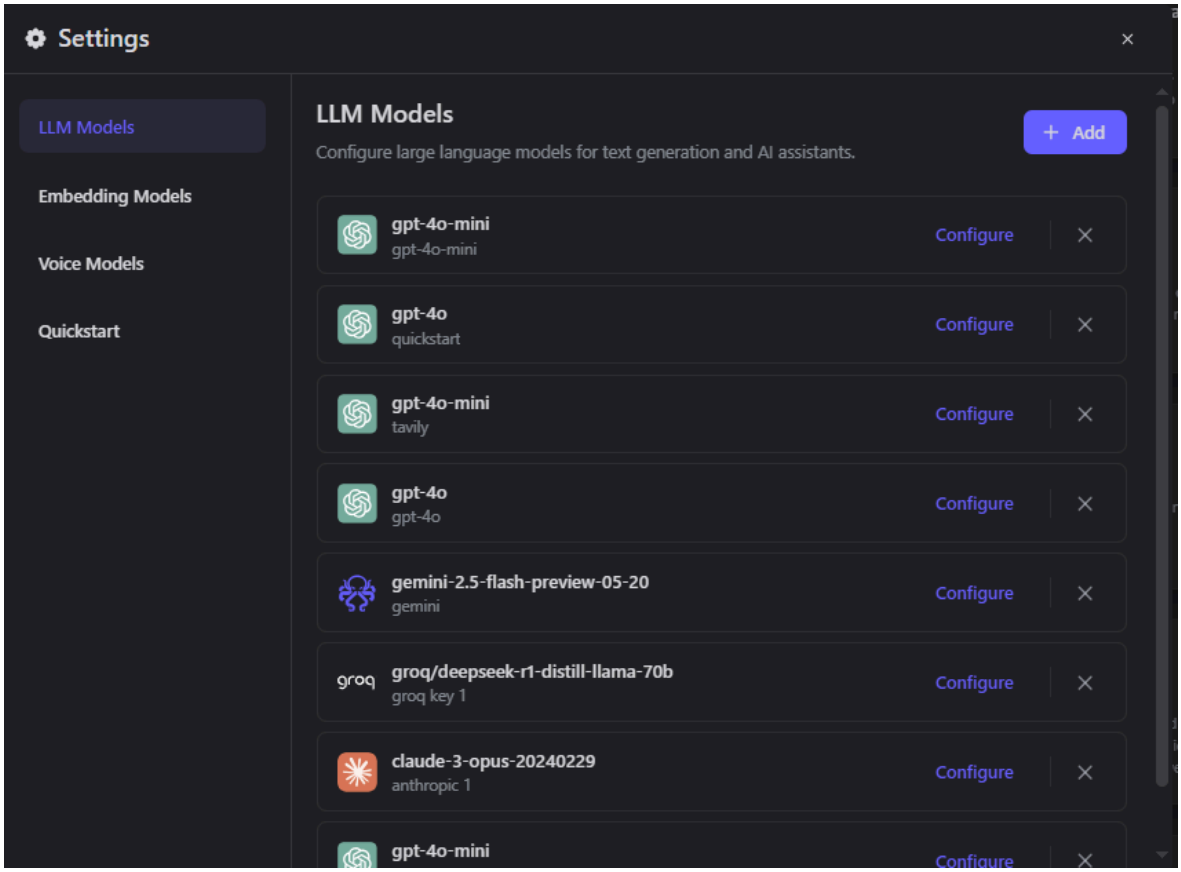
Before you can use agents, flows, tools, or any intelligent features — you **must** set up your LLM models (Large Language Models). This is a required step.

## Where to find it

1. Go to the **bottom-left corner** of the screen.
2. Click the **gear icon** (⚙️ **Settings**).




3. A **modal window** will appear with **4 tabs**:
   - `LLM Models`
   - `Embedding Models`
   - `Voice Models`
   - `Quickstart`




These tabs are where you **create and manage all the models** needed in your system.

---

# LLM Models tab

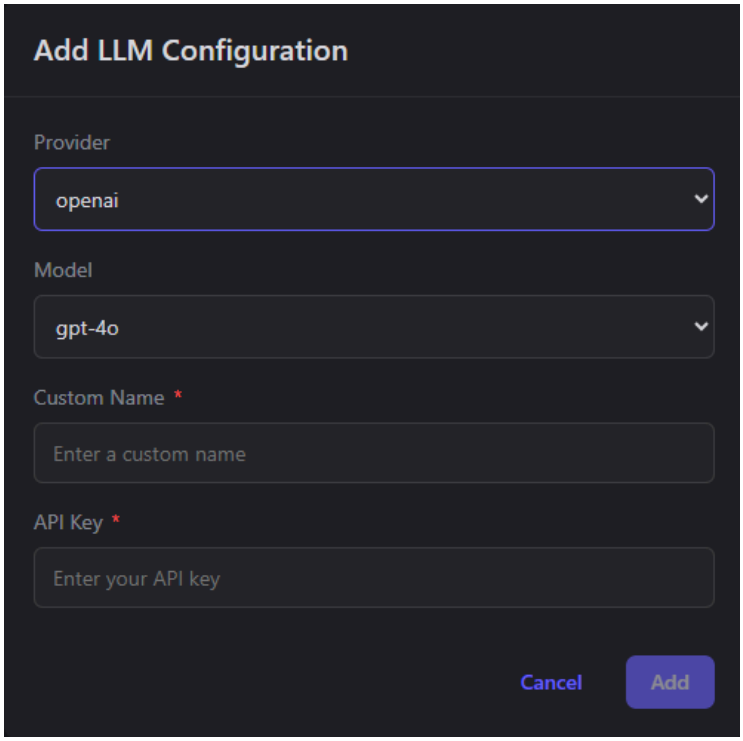
## How to Add a New LLM

1. Click the **`+ Add`** button on the `LLM Models` tab


2. A modal will open with the following required fields:

    - **Provider** – Select from a dropdown (e.g., OpenAI, Anthropic, etc.)
    - **Model** – Choose the model name (options depend on the selected provider)
    - **Custom Name** – Give your model a name (you'll use this name later)

- **API Key** – Paste your provider's API key




3. After filling out everything, click **Add.** The model will appear in the list of all available LLMs.

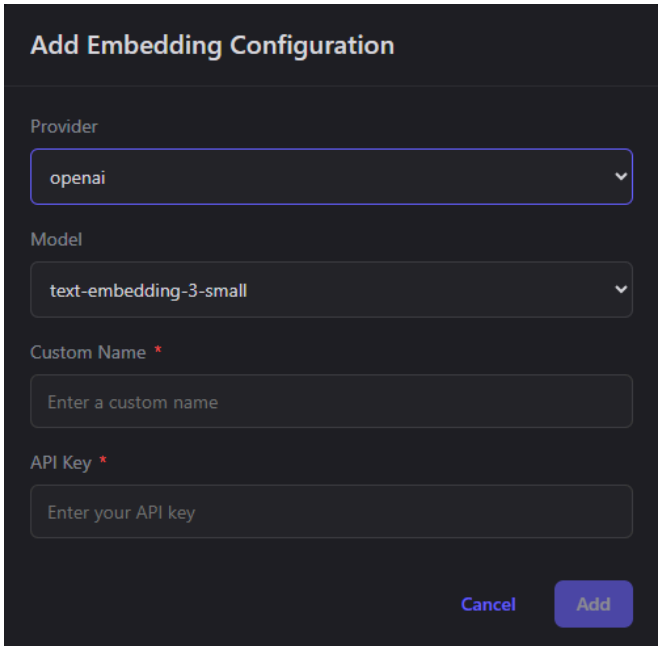
---

## Embedding Models tab

This tab works **exactly like LLM Models**, but for **embedding models** (used for memory, semantic search, etc.).

To add one:

- Click `+ Add`
- Fill out: Provider, Model, Custom Name, API Key



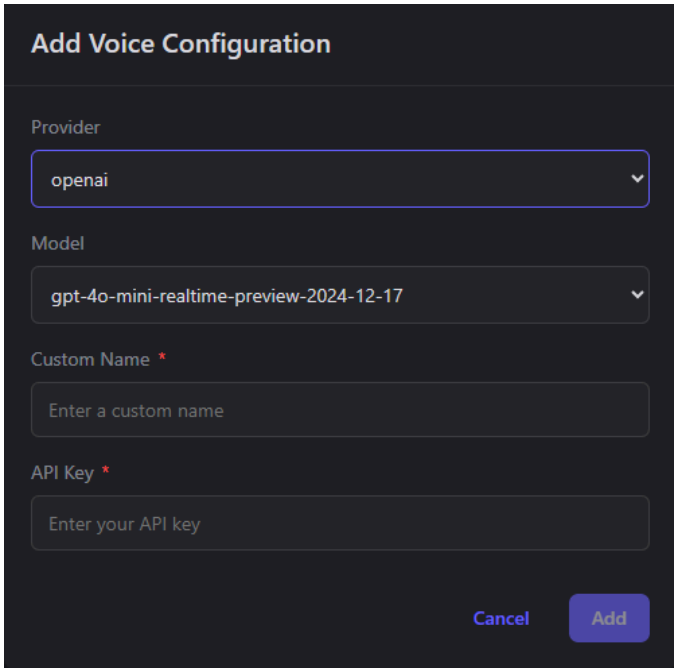

- Save → The model appears in your list

---

## Voice Models tab

Again, this tab works the same way. Voice models are used for **speech synthesis** (e.g. if your agent speaks aloud in a real-time chat).

Just click + Add and follow the same steps.

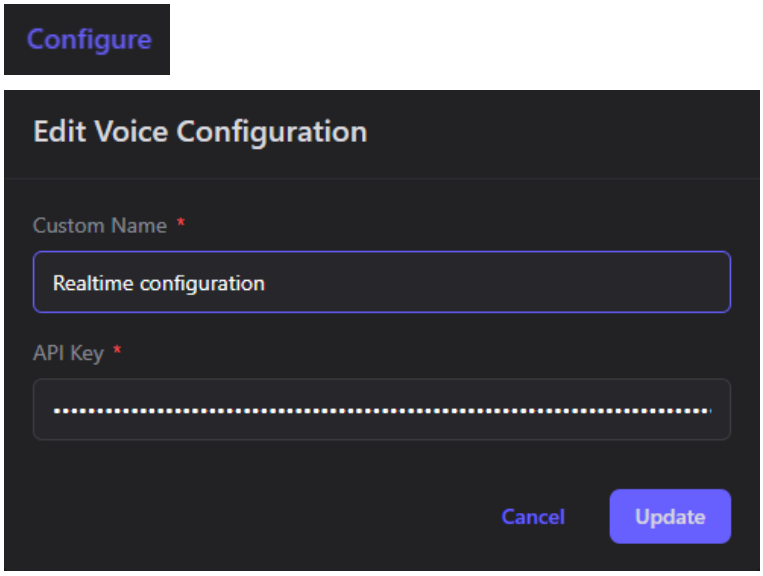



---

## What you can do with each model in the list

Once you've added models, you can:

- **Edit / Configure** (change name or API key)

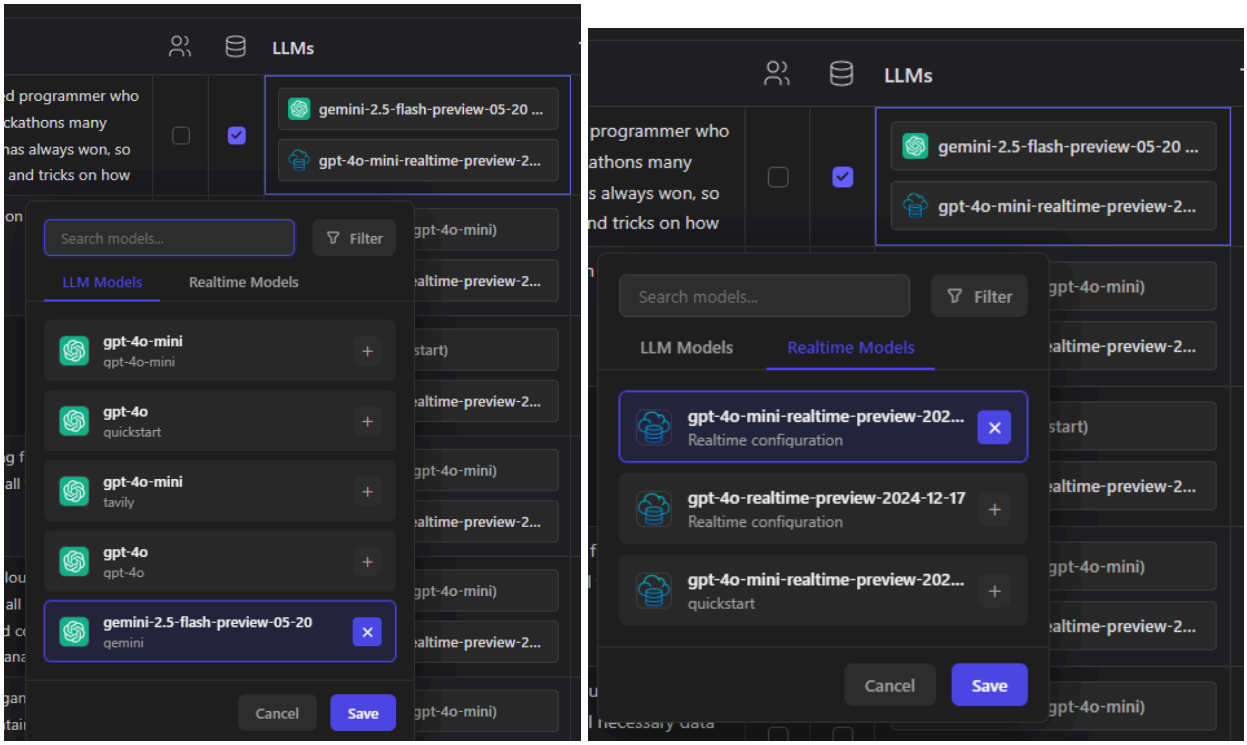



- **Delete** (remove from the system)

## How to assign a model to an agent

To make agents actually use a model:

1. Go to the **Staff page** (where your agents are listed)
2. Find the cell under the column for `LLM Model` or `Realtime Model`
3. **Double-click** the cell
   A small window will appear → select the model you want to assign




## Quickstart Setup (OpenAI)

The **Quickstart** tab in the Settings panel helps you rapidly configure your environment using a single OpenAI API key.

### What It Does

By entering your **OpenAI API Key**, the system will **automatically create and configure** everything you need to start building immediately:

- LLM models and tool configurations
- Realtime model settings
- Embedding model settings
- Default models for:
    - Projects
    - Agents
    - Tools

This is the **fastest way to get started**, especially useful for first-time users or quick testing.

### How to Use It

1. Paste your **OpenAI API Key** into the input field.
2. Click **“Start Building”**.
3. The platform will pre-fill all necessary model and tool settings for you.

⚠️You can still manually adjust or expand these configurations later under the `LLM Models`, `Embedding Models`, or `Voice Models` tabs.

---

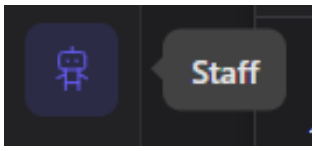
## Why This Is Important

Nothing in your system will work without assigning models. Flows will not run, agents will not respond, and real-time chats won’t function unless:

- At least one LLM model is added
- It is enabled and properly assigned to agents

Staff in EpicStaff are your workers, agents — virtual helpers, who will bring to life any idea you have. This is where you define the intelligent agents that will perform tasks in your project.

When you open the Agents page, you'll see a table listing all the agents in your project. Each **row** represents a separate **agent**. Each **column** contains a specific property of the agent, which you can edit directly in the table by **doubleclicking** the cell you want to modify.




There are **two ways to create and manage agents**:

1. Using the **Table View**, which allows quick editing of multiple agents in a spreadsheet-like format.
2. Using the **Form View**, which provides a step-by-step interface for creating or editing one agent at a time.

## 1. Creating Agents via the Table View format

Below is a breakdown of each column and its purpose:

### Agent Role

*What it is:* A short name or title that describes the function of the agent (e.g., "English Teacher", "Trip Planner", "John Swift").

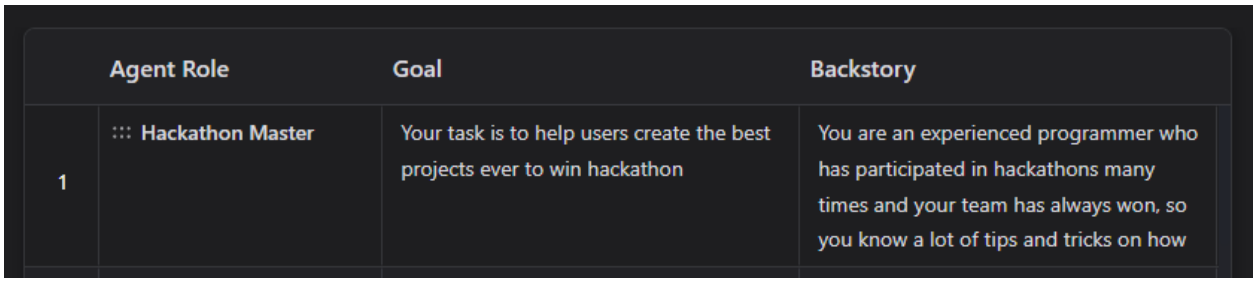
### Goal

*What it is:* A text describing **what this agent is supposed to accomplish**.
*Example:* “Search for the cheapest flights that match the user’s preferences.”
*Tip:* This helps the system understand the agent’s purpose and align tasks accordingly.

### Backstory

*What it is:* A narrative-style background for the agent.
*Why it matters:* This helps the system build more human-like, context-aware reasoning by imagining the agent as a “character.”

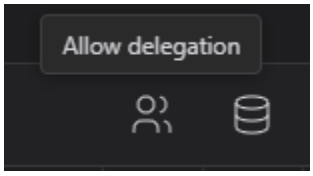
*Example:* “A former travel agent who now uses AI to help people plan trips faster and smarter.”



| | Agent Role | Goal | Backstory |
|---|---|---|---|
| 1 | ::: Hackathon Master | Your task is to help users create the best projects ever to win hackathon | You are an experienced programmer who has participated in hackathons many times and your team has always won, so you know a lot of tips and tricks on how |

## Allow Delegation

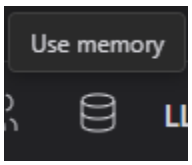
*What it is:* A checkbox that enables the agent to delegate tasks to other agents.
*How to use it:* If checked, the agent can decide to pass on subtasks to other agents in the system or call other agents for help while performing its task.
*When to enable:* Useful for agents responsible for complex tasks (e.g., “Trip Planner” can delegate hotel search to another agent).




## Memory

*What it is:* A checkbox that enables the agent to store all information it gathers or generates during a single flow run. This memory is accessible to the user after execution and can be reviewed or deleted as needed.

To find out more about memory go here: Memory

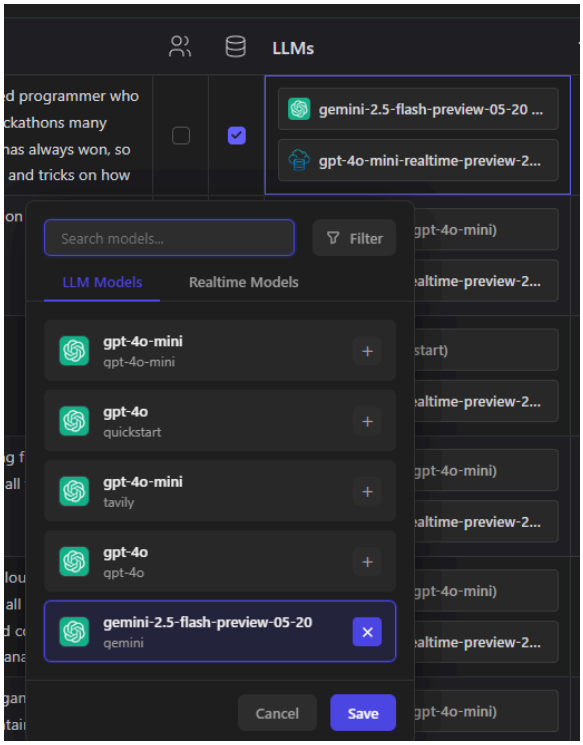


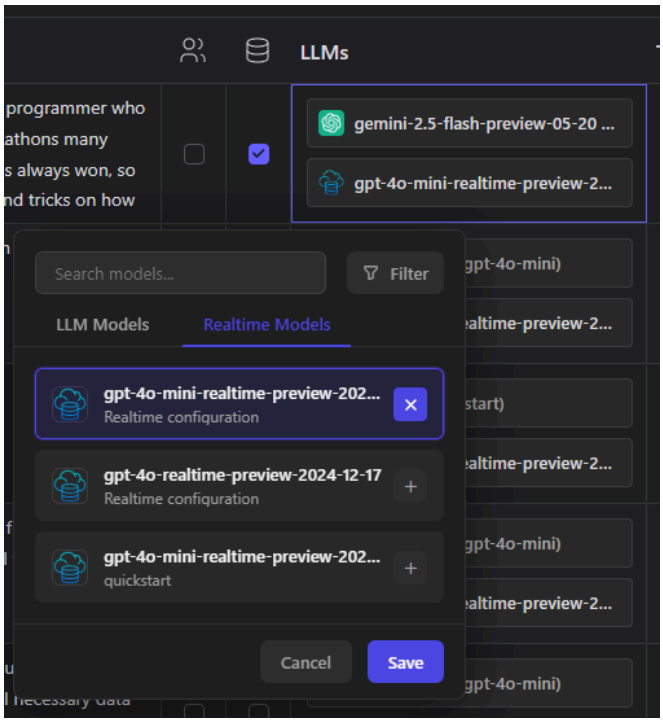
## LLMs (Language Models)

This column shows which AI models are assigned to the agent. It may contain two subfields:

- LLM Models – the main language model (e.g., GPT-4, Claude) used to process tasks for this agent.
- Realtime Models — optional models that enable **real-time voice conversations** with the agent.

To find out more about setting up LLMs go here: [LLMs configurations]



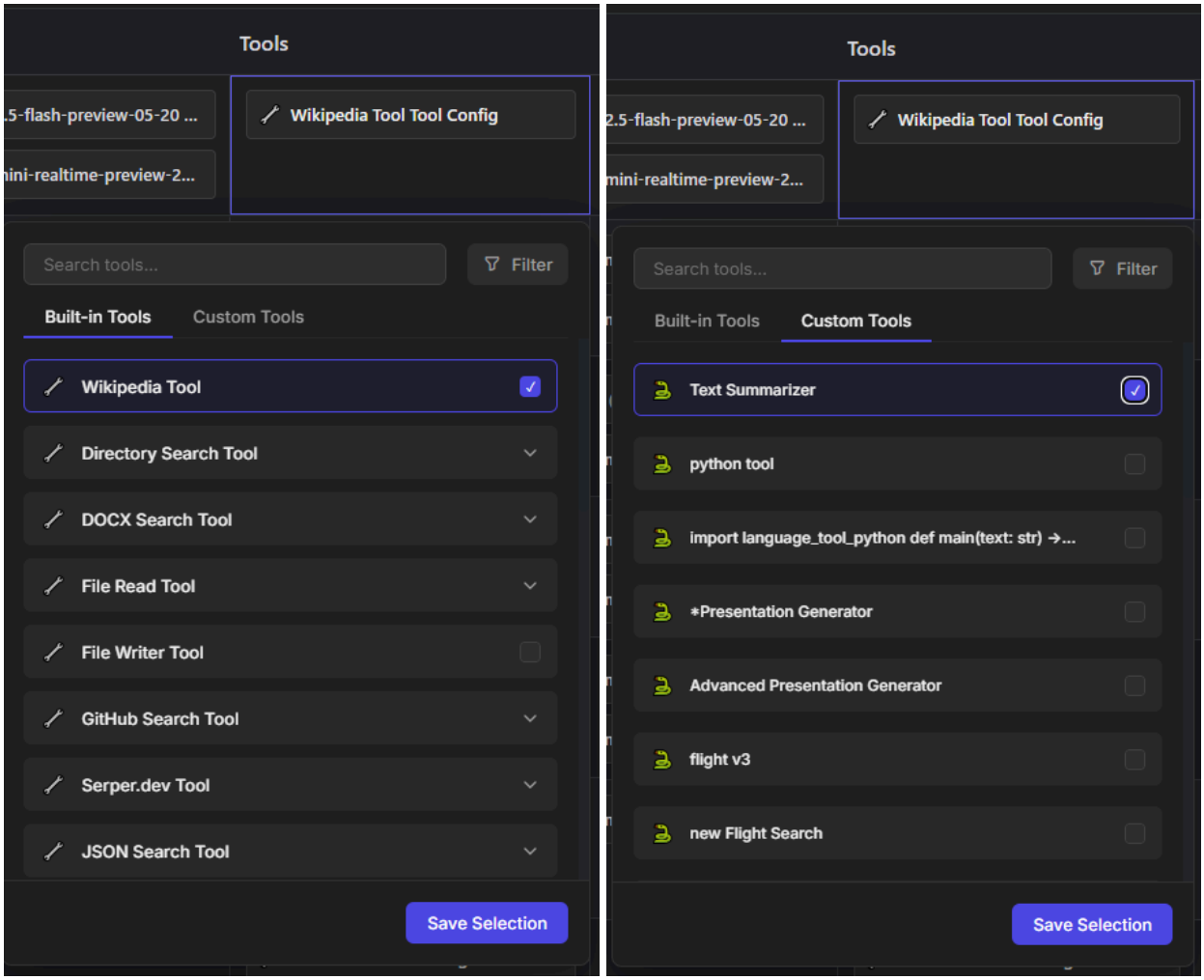

## Tools

This column defines the **capabilities** the agent can use to perform its tasks.

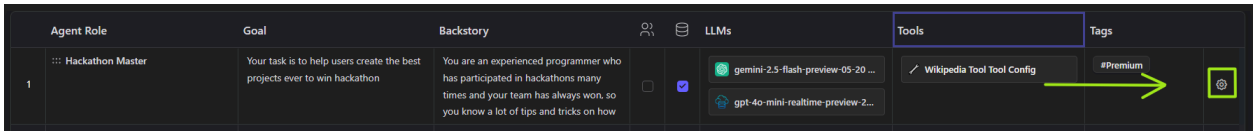
- Built-in Tools — tools that are part of the system by default (e.g., wikipedia, dall-e).
- Custom Tools — Python-based tools created by the user. These can be added to give the agent access to custom logic, APIs, or special actions.

*How to use:* You can assign one or more tools to an agent using a dropdown or tool selector. Make sure the tools are relevant to the agent's role and goal.

Read how to create and manage tools here: Tools




At the end of each agent row, you'll find a **gear icon** ⚙️. Clicking it opens a panel with **advanced configuration options** for the selected agent. These settings allow fine-tuning of how the agent behaves during execution.

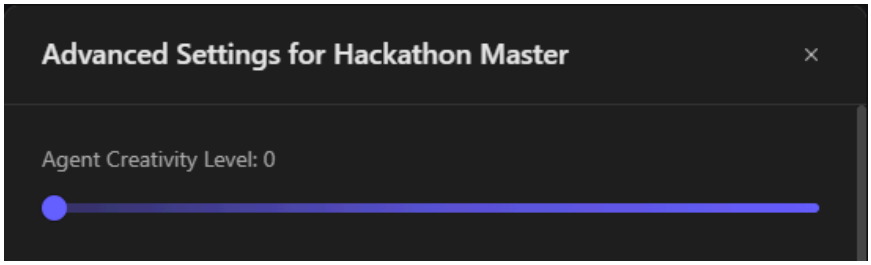



Here's what you'll find inside:

## Agent Creativity Level (1–100)

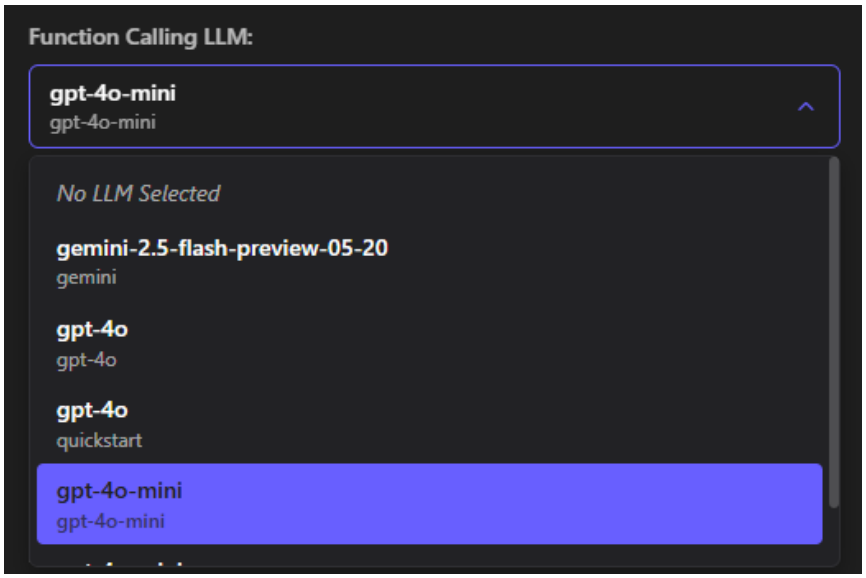
Controls how imaginative or focused the agent's responses are.

- ➔ Lower values = more precise, conservative answers, does only what you ask of it.
- ➔ Higher values = more creative, open-ended thinking.




## Function Calling LLM

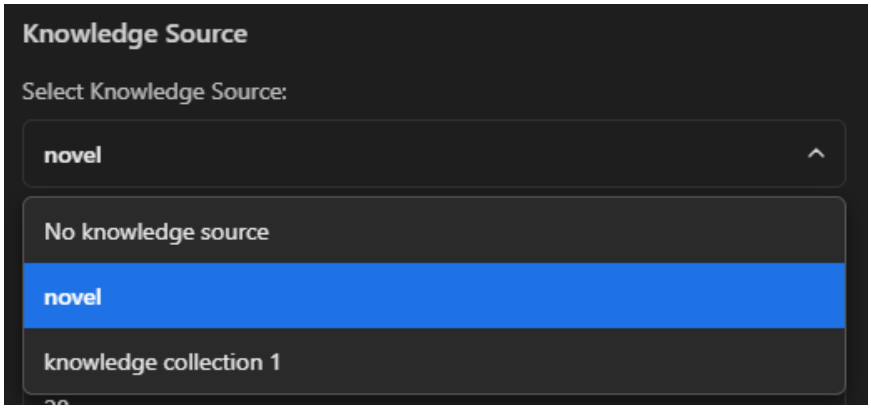
Select which language model should be used specifically for **function/tool calling**. Useful if you want to separate general reasoning from function execution.




To find out more about setting up LLMs go here: LLMs configurations

## Knowledge Source

Choose the data source the agent should rely on during execution — e.g., web search, internal docs, or no external knowledge.
Read how to create knowledge source here: Knowledge




## Execution Settings

Fine-tune how the agent operates during a flow:

**Maximum Iterations**: Sets the maximum number of steps the agent can take to think, use tools, and generate results during one flow run.

**Maximum Requests Per Minute**: Throttles how often the agent can call tools or APIs.

**Maximum Execution Time (seconds)**: Time limit for completing its tasks.

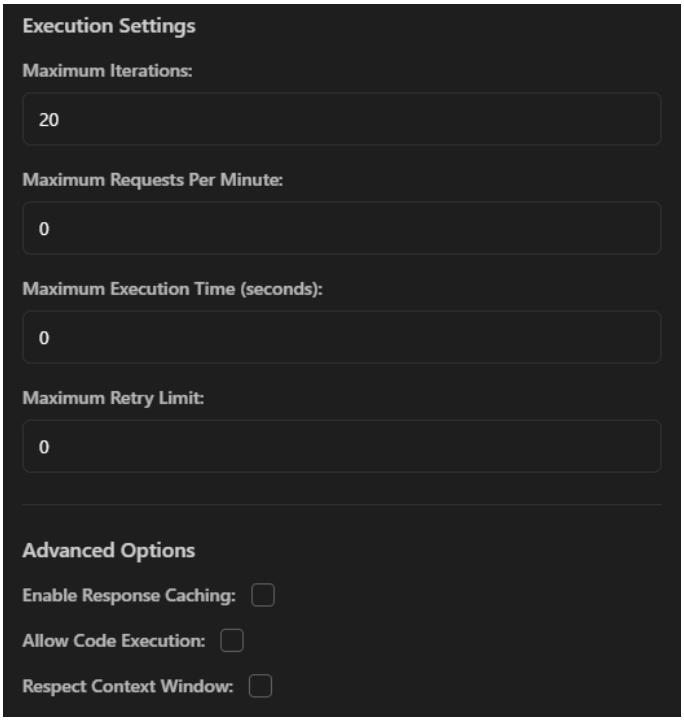
**Maximum Retry Limit**: How many times the agent will retry if something fails.

## Advanced Options

**Enable Response Caching**: Caches the agent's responses to avoid repeating expensive operations.

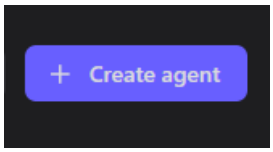
**Allow Code Execution**: Allows the agent to run Python or other executable code when needed.

**Respect Context Window**: Forces the agent to stay within the context limit of its model (prevents overly long responses or memory overflows).




## 2. Creating an Agent via the “Create Agent” Modal

Another way to create an agent is by clicking the **“Create Agent”** button in the top right corner of the page. This opens a **modal window** with a form for entering the agent’s settings.




The form includes **almost the same fields** as in the table view — both the main properties and the advanced settings (normally hidden behind the gear icon).

Here are all the fields available in the modal:

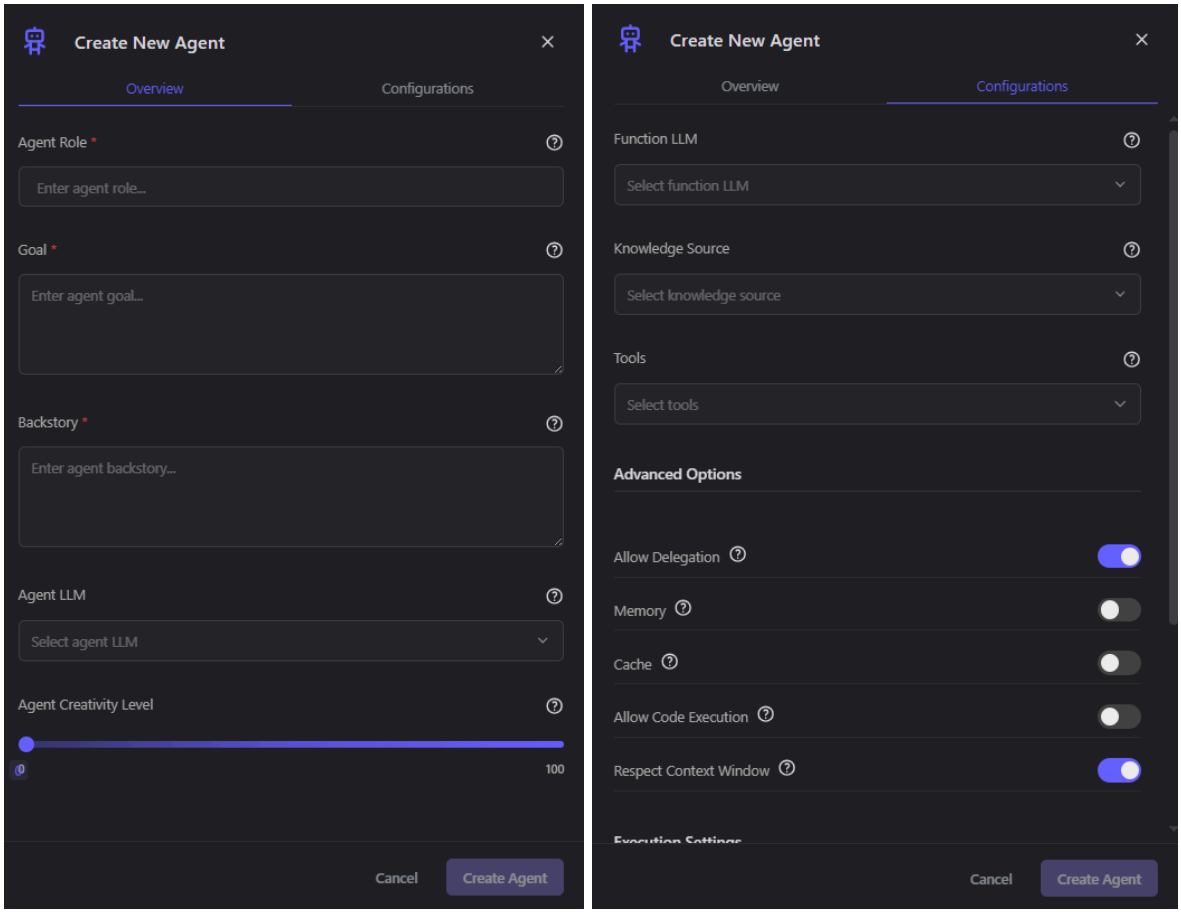
- Agent Role
- Goal

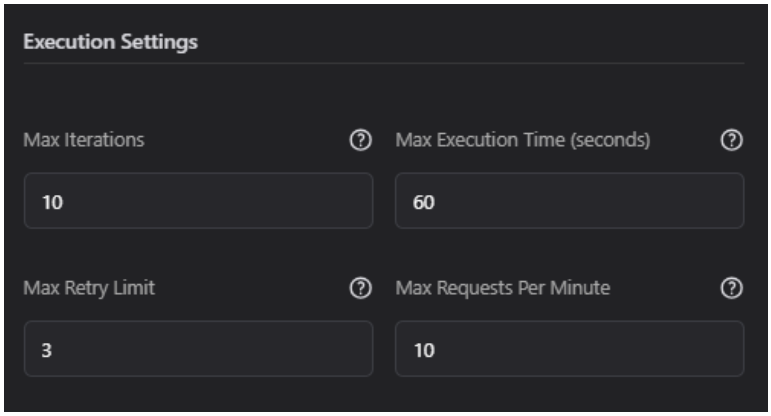
- Backstory
- Agent LLM
- Agent Creativity Level

## Advanced Settings section

- Function LLM
- Allow Delegation
- Memory
- Cache *(Enable Response Caching)*
- Allow Code Execution
- Respect Context Window
- Maximum Iterations
- Maximum Execution Time (seconds)
- Maximum Retry Limit
- Maximum Requests Per Minute

After filling out the fields, click **“Create”** to add the agent to your project.






---

## ⚠️Notes on Creating Agents

To successfully create an agent, the following fields are **required**:
**Agent Role**, **Goal**, and **Backstory**.

To ensure the agent can actually perform tasks in a flow, you also need to assign an **LLM model** (under **Agent LLM**).

All other fields are optional and can be customized later.

---

## Table Actions (Right-Click Menu)

When working in the Table View, you can **right-click** on any row to access quick actions:

- **Add Empty Agent Above / Below**
  Inserts a new blank agent row directly above or below the selected one.

- **Copy Row**
  Copies the selected agent row (including all its settings).

- **Paste Row Above / Below**
  Pastes the previously copied agent into the table above or below the current one.

- **Delete Row**
  Removes the selected agent from the table.

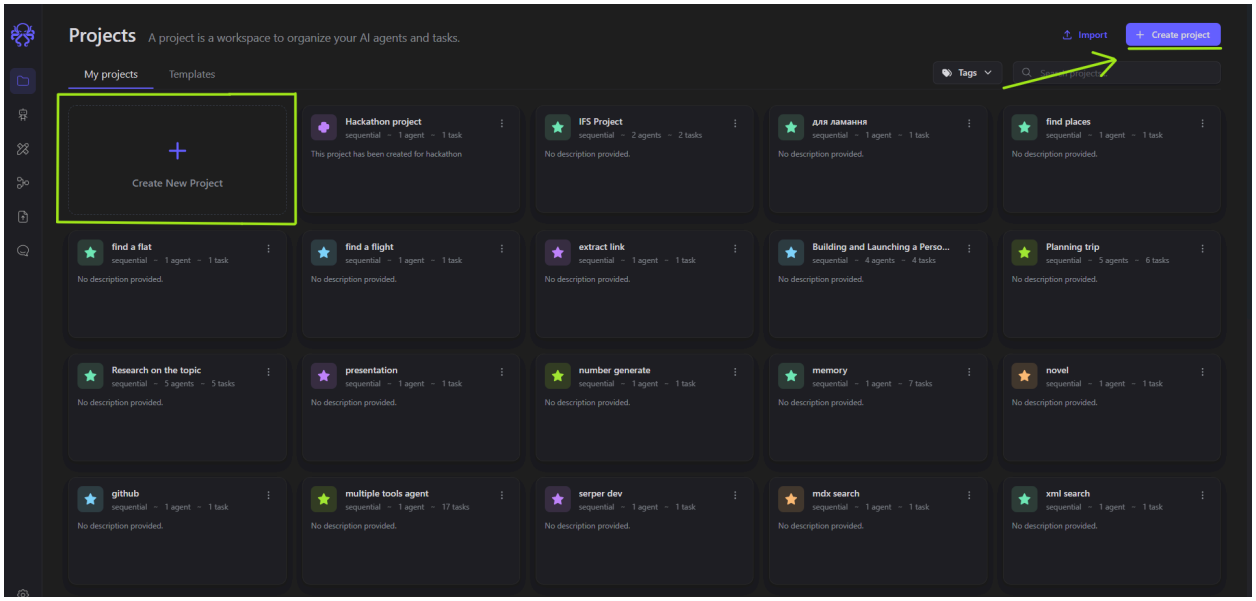
A **project** is the starting point for building your flow.

It groups together all the core elements — agents, tasks, and logic — into a single workspace where everything works together.

Before you can create agents or tasks, you need to start by creating a project.

## Creating a Project

You can create a new project by clicking either of the two **“Create Project”** buttons available on the Projects page.
*(Both buttons open the same form — the choice doesn’t matter.)*




Once clicked, a **modal window** appears with the following fields:

### Main Fields:

**Project Name** *(required)* – The unique name of your project.

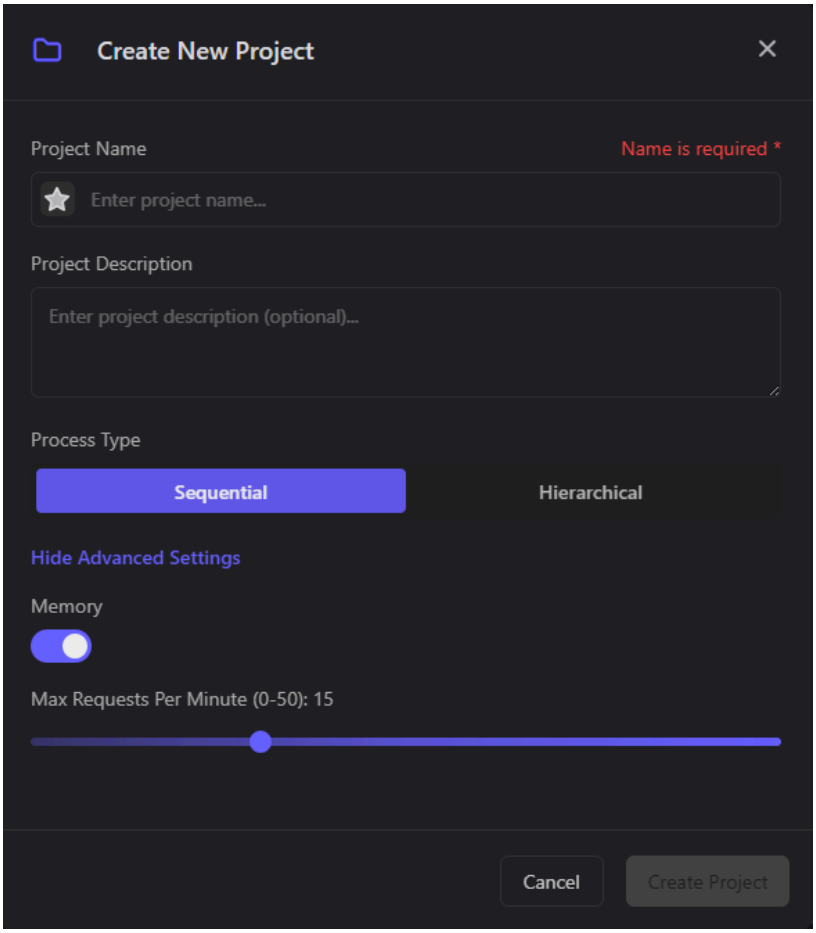
**Project Description** – Optional text to explain the purpose or scope of the project.

**Process Type** – Choose between:

- **Sequential** *(currently the only supported mode)*
- **Hierarchical** *(UI option only — not yet active)*

### Advanced Settings:

**Memory** – Enable or disable short-term memory during flow execution.
**Max Requests Per Minute** – Throttle agent tool/API calls across the project (range: 0–50)




After filling in the form, click **“Create”** to generate the project.

---

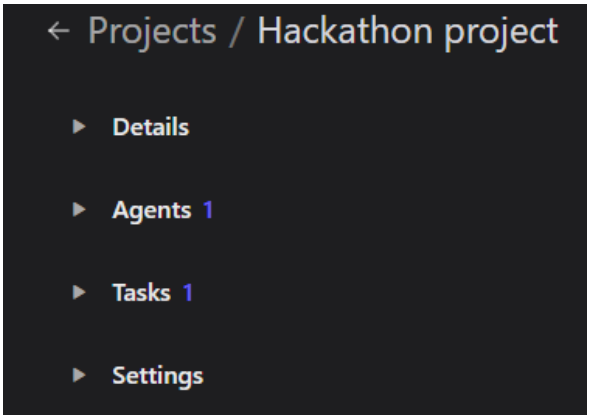
## Project Main Page

Once your project is created, you’ll be redirected to its **main page**, which is divided into several key sections:

- **Details** – General information and editable project properties.
- **Agents** – The agents participating in this project.

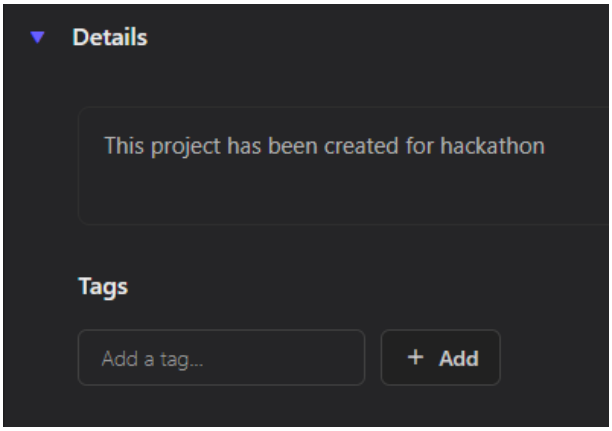
- **Tasks** – The tasks (steps) that define your flow logic.
- **Settings** – Advanced execution parameters and options.

Each section can be edited individually and contributes to how the flow will behave during execution.




## 1. Details

Here you can **view or edit the project description**. Click inside the description field to update the text and provide more context about your project.




## 2. Agents

This section lets you add agents to the project from the system-wide list.

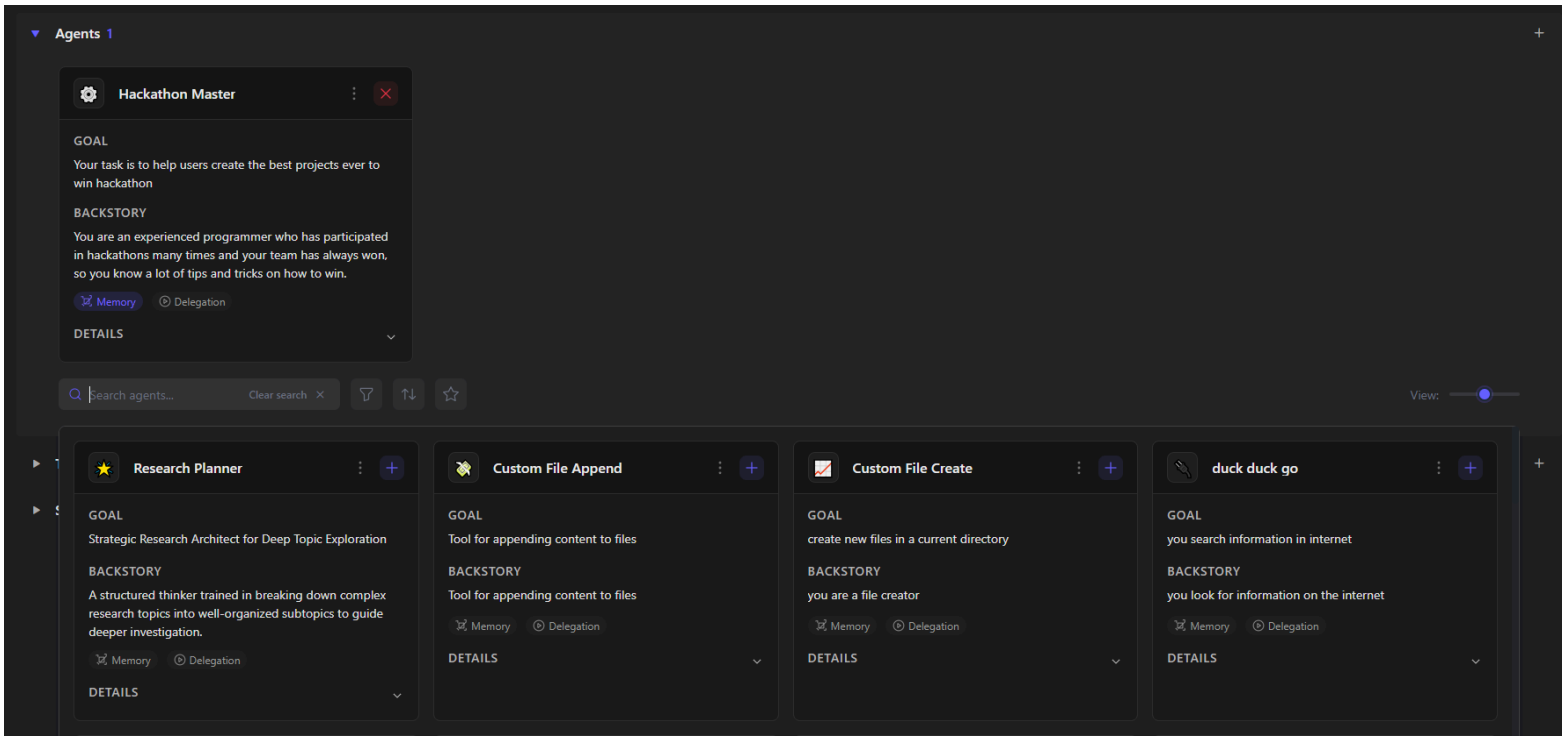
How to add agents:

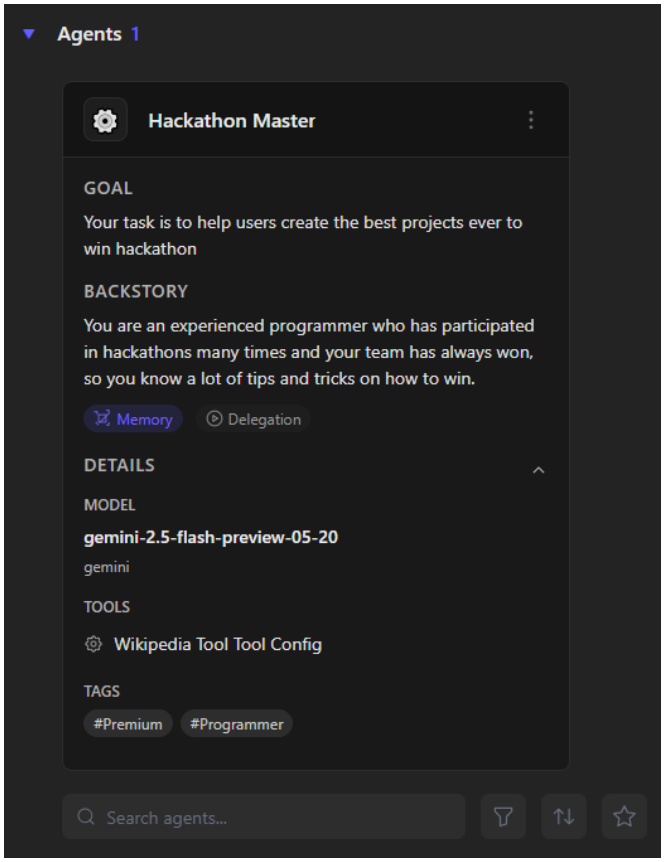
1. Click the **“Search agents”** field.
2. A list of all existing agents will appear.
3. Type the name (or part of the name) to filter the list.
4. Click the ➕ plus icon in the top-right corner of the agent’s card to add it.

Once you’ve added all the needed agents:

- You can click **“Clear search”** to hide the full agent list.
- Added agents are shown as cards with key info.
- To view full details of an agent, click **“Details”** on the card.
- To remove an agent, either:
  - Click the ❌ **cross** in the top-right corner of the card, or
  - Click the **three dots** → **Delete**

Read how to create and configure agents here: [Staff]






## 3. Tasks

Tasks define what each agent in the project should do. They are created and edited using a spreadsheet-style table.

### Required fields to define a task:

- **Task Name** – A short label.
- **Instructions** – What the agent should do.

This is the **core prompt** for the agent. Use it to clearly describe the **task or action** the agent is supposed to perform.

**Good instructions are:** clear and specific, focused on one task at a time

### Examples:

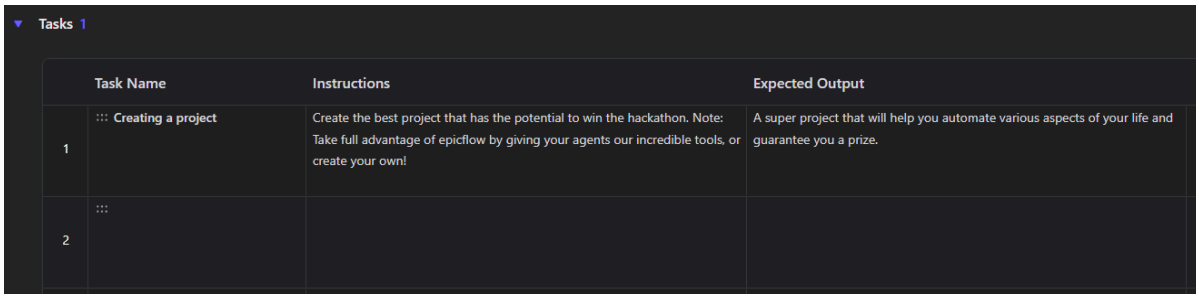
“Find 3 affordable flights from Berlin to Lisbon in early July.”
“Summarize the main points from the article provided.”
“Generate a list of blog title ideas for a post about digital detox.”

- **Expected Output** – What kind of result is expected.

This is where you **define the format or structure** of the agent's answer.
It helps the agent understand how to present the result and also makes it easier for you to use that result later in the flow. You can describe the expected output in natural language, or define it in a more structured way (e.g., JSON).

**Examples:**

"A short paragraph summarizing the article."
"A numbered list of three options, each with name, price, and link."
"JSON with fields: destination, price, airline, departure_time."



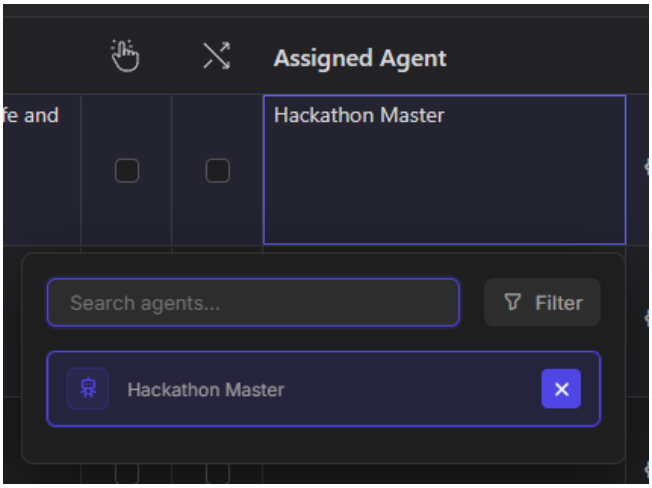

⚠️ **Important:**

Each task **must be assigned to an agent** from the project.
Tasks **cannot run without an agent**, but agents without tasks are allowed.

**Assigning an agent to a task:**

1) Double-click the **agent cell** in the row of the desired task.
2) Select an agent from the dropdown list (it only shows agents added to the project).

Find out more about how to pass variables through tasks here: Variables

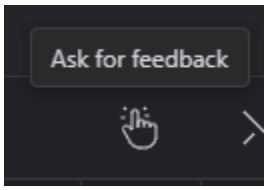



**Additional options per task:**

Each task row includes two checkboxes:

- **Human Input** – If checked, the flow will pause and wait for user input at this step.

Read more about Human input here: [Human Input]



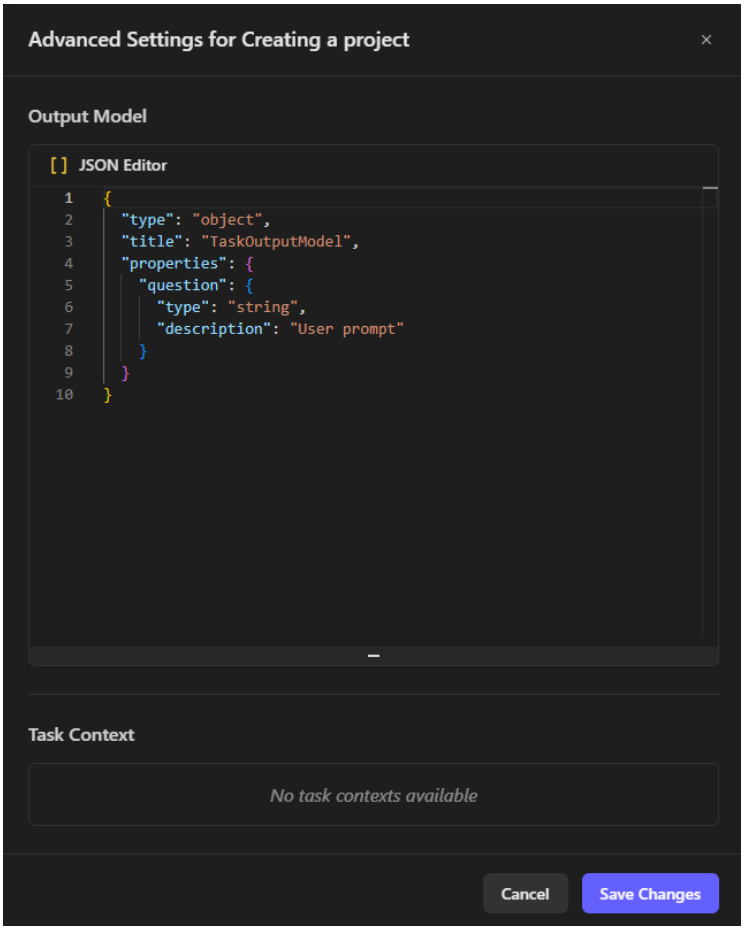

**Advanced task settings:**

Click the **gear icon** at the end of a task row to open **Advanced Settings**. There you can specify an **Output Model** in JSON format.

In the next picture is an Output Model of the task that LLM will generate. For example, if we want a JSON as output like this

```
{
  “question”: “some text”
}
```

Then we need to specify the properties of the schema, there must be a field with a name **question**. Inside this object there should be a **description** for LLM about the field content and **type** of this field (for example **string**).
There may be any amount of fields, but in example there is only one.

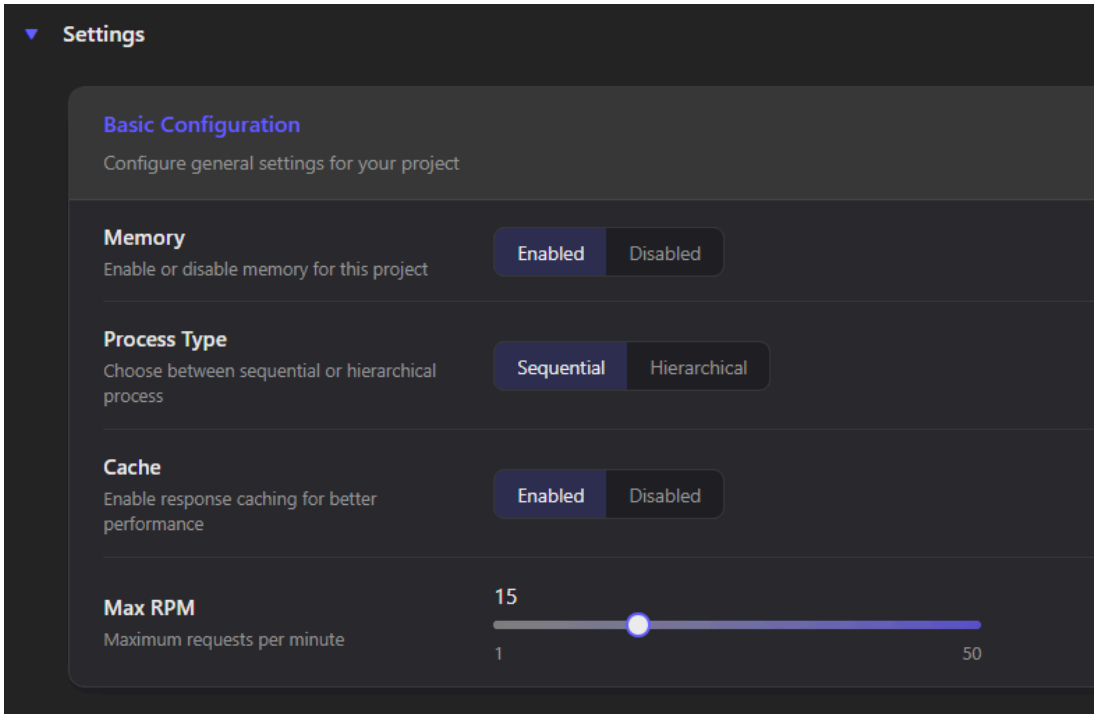



**Note**:
If the **last task** of the project contains the **Output Model**, then the result of the **project node** will be the same format as described in the last task.

## 4. Settings

This section lets you configure project-level execution options:

- **Memory** – Enable/disable short-term memory during flow run. You can read more about memory here: Memory
- **Process Type** – Choose between **Sequential** (supported) or **Hierarchical** (not yet supported).
- **Cache** – Enable response caching.
- **Max Requests Per Minute (1–50)** – Controls the request limit across all agents during execution.


Settings
Basic Configuration
Configure general settings for your project
Memory
Enable or disable memory for this project
Enabled
Disabled
Process Type
Choose between sequential or hierarchical process
Sequential
Hierarchical
Cache
Enable response caching for better performance
Enabled
Disabled
Max RPM
Maximum requests per minute
15
1
50

# What are tools — and why do agents need them?

**Tools** are external functions that agents can call during execution to do something beyond what the language model (LLM) can do on its own. Think of them as **plugins** or **extra capabilities** that extend the power of the LLM.

## LLM vs. Tool:

- The **LLM** (Language Model) is great at:
  - Generating text
  - Understanding language
  - Answering general questions based on training data
  - Following instructions

- But the LLM **cannot**:
  - Make API calls or get real-time data
  - Do complex math or code execution
  - Read files
  - Search the web
  - Interact with databases

That's where **tools** come in.

## Types of tools in the system

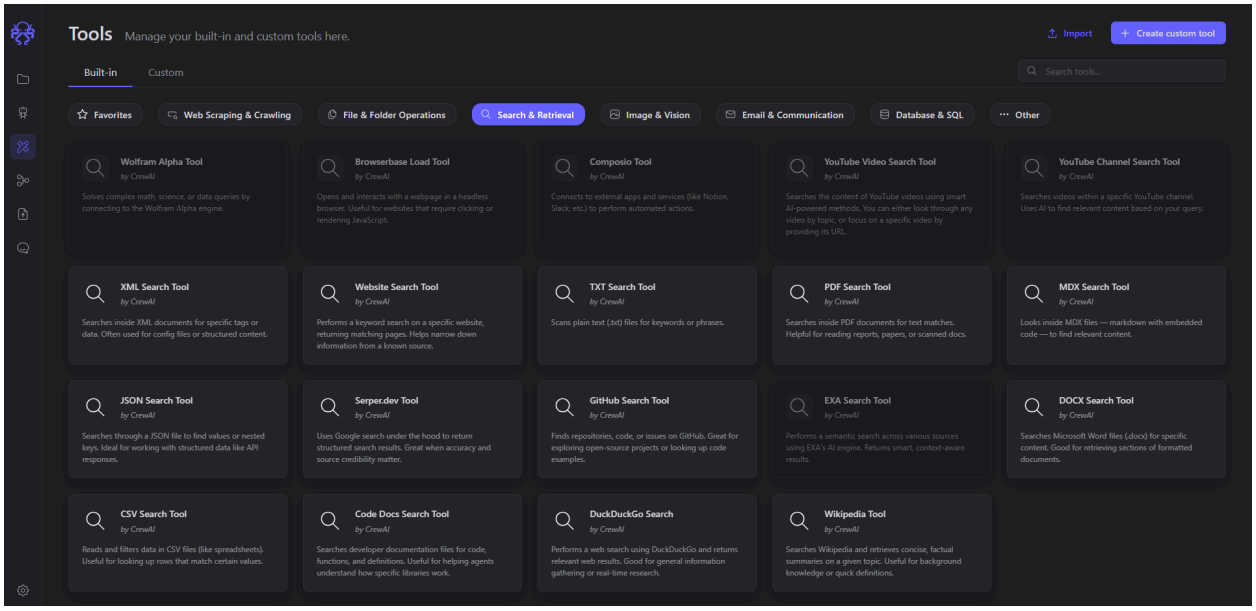
There are **two kinds** of tools in the system:

1. **Built-in tools** – already integrated (e.g., Google Search, Calculator, Web Scraper)
2. **Custom Python tools** – created by you or your team for specific tasks (e.g., CSV converter, database connector)

# Built-in Tools

Built-in tools are ready-made tools available in the system. They are grouped by categories (e.g., Search, Text Processing, Communication, etc.) and can be attached to agents to extend their capabilities.

Here you can find a detailed list of tools that are working: Tools

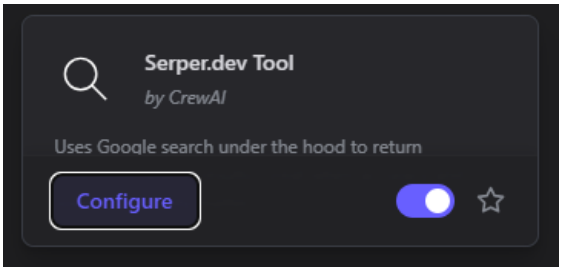



## Tool Configurations

Some tools require additional setup through configurations.

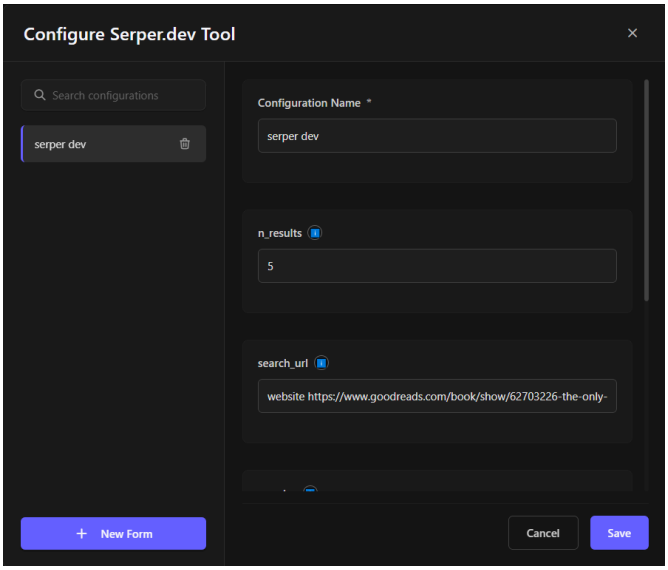
### How to configure a built-in tool:

1. Hover your mouse over the tool.
2. Click the **“Configure”** button that appears.
3. A configuration form will open




### The configuration form includes:

➢ **A name for the configuration** (required for all tools)
➢ **Additional parameters** (varies by tool)




⚠️Fields with a **blue info icon** have tooltips — hover to read a short explanation.

⚠️Fields marked with **asterisks (*)** are required to enable the **Create** button.

### To add a new configuration:

1. Click **+ New Form**
2. Fill in at least  the required fields (those marked with *).
3. Click **Create**.

### To edit an existing configuration:

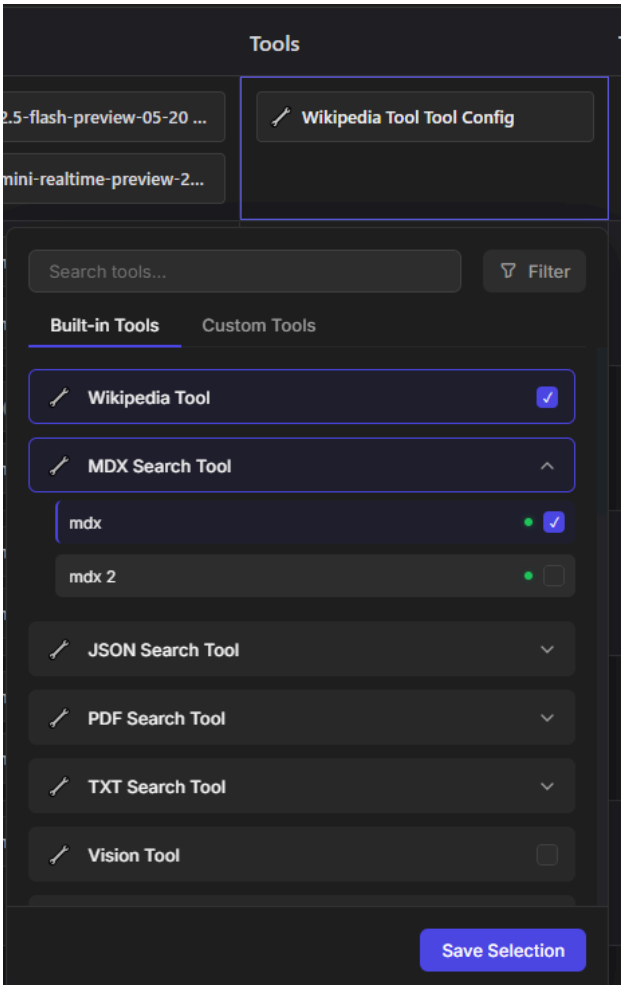
1. Select it from the configuration list.
2. Make your changes.
3. Click **Save**.

---

## Assigning a built-in tool to an agent

1. Go to the **Agents** section of your project.
2. Double-click the cell under the **Tools** column for the desired agent.
3. In the popup window, switch to the **“Built-in”** tab.
4. Click on the tool you want to assign.
5. If the tool has at least one saved configuration, a **dropdown menu** will appear.

6. Select the configuration by name — and it will be assigned to the agent and save the selection.



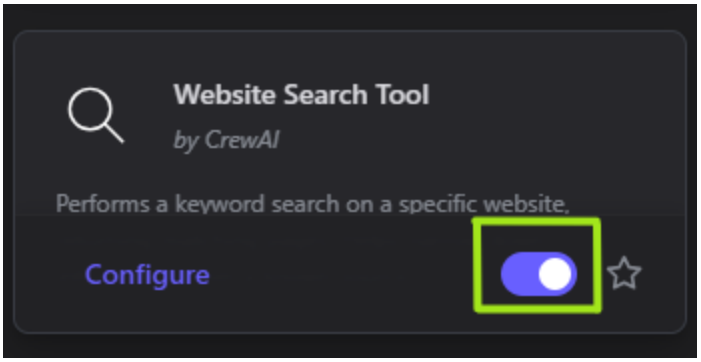

---

## Disabling Tools

You can temporarily **disable** any tool by using the toggle switch that appears when you hover over it.

- When toggled off (grayed out), the tool becomes inactive.
- **Disabled tools won't appear** in the tool selection window when assigning tools to agents.

This is useful if a tool is deprecated, not needed for now, or under testing.



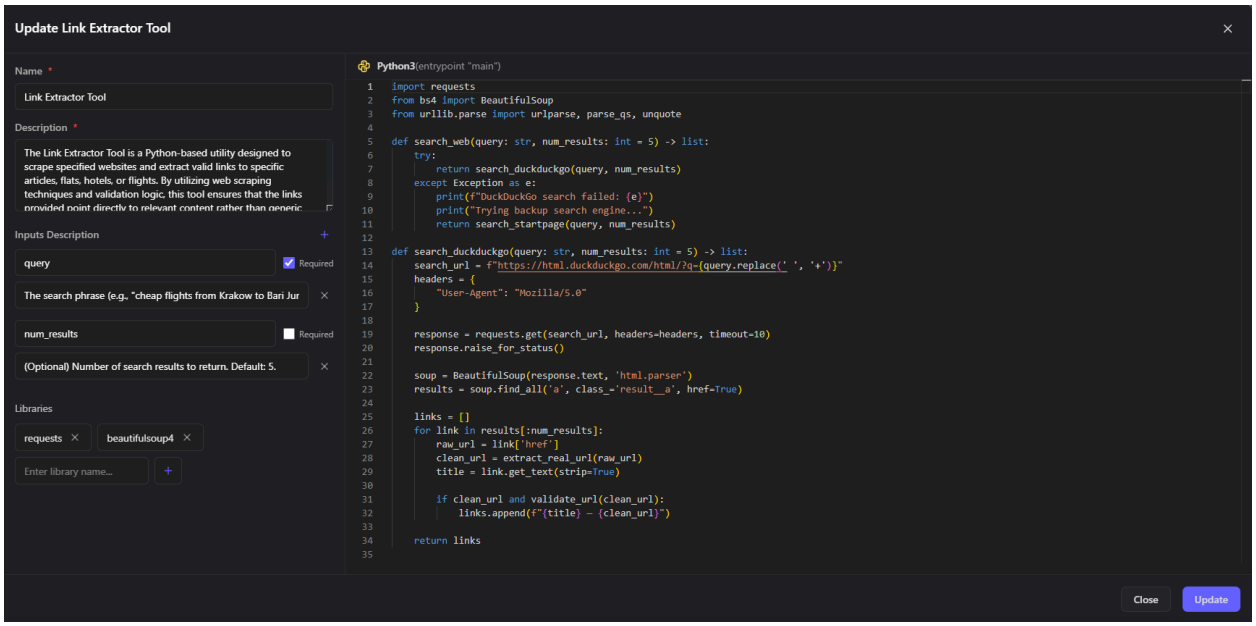

---

# 🐍 Custom Tools (Python)

Custom tools allow users to build their own Python-based functions and assign them to agents. These tools are helpful when built-in options are not enough for specific logic or data handling.

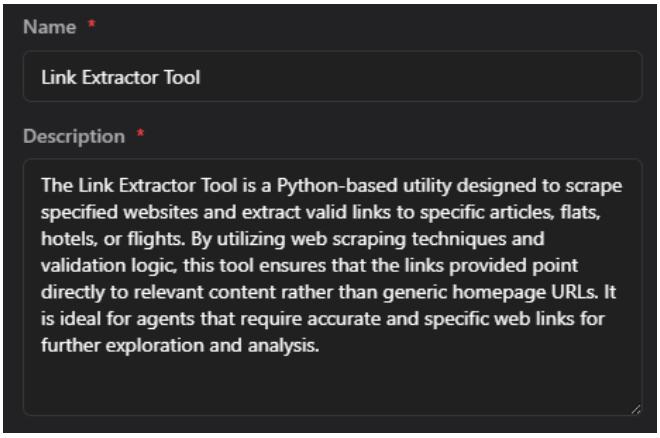
## How to create a custom tool

1. Go to the **Tools** page and click the **“Create Custom Tool”** button in the top right corner.
2. A modal window will open with the following fields:




## Required Fields:

- **Name** – unique name for your tool
- **Description** – short explanation of what the tool does




## Inputs Description:

- Click the **+** icon to add input parameters to the `main()` function
- For each input, provide:
    - **Input Name** – must match the parameter name in your Python code.

*What it is*: The **Input Name** is the name of a parameter that your custom Python function (`main()`) expects. It **must exactly match** the parameter name in your function signature.

*Example*: If your function is:

```
def main(city: str, date: str) -> dict:
```

You must define two inputs with names:

- `city`
- `date`

These must match **exactly** — case-sensitive, no typos.

- **Description**

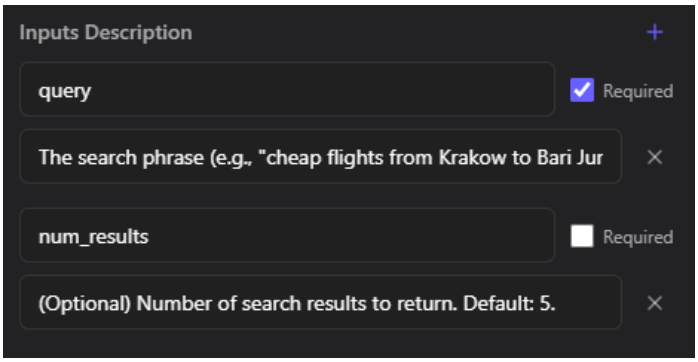
*What it is:* A short explanation of what the input represents or how the user should fill it in.

Example Descriptions:

- For `city`: *"Destination city for flight search"*
- For `date`: *"Preferred departure date in YYYY-MM-DD format"*

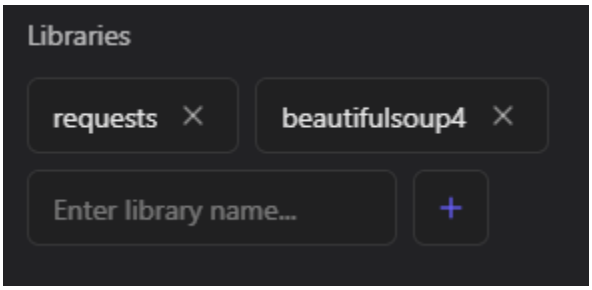
Why it's useful: The description shows up as a tooltip (info icon □), helping other users understand what each input is for when they configure or use the tool.

    - Mark as **Required** if the tool won't work without it




## Libraries:

- If your code needs external libraries, enter them in the **"Enter library name…"** field
- ⚠️ Press **Enter** or click the + button to confirm
- ⚠️ *You don't need to add standard Python libraries manually — some are already included in the environment.*

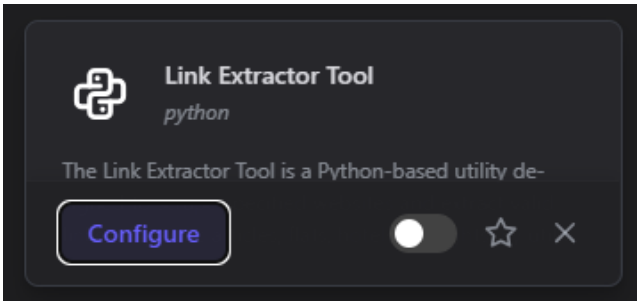



### Python Code Editor:

- Define a single Python function:
- The function:
    - Must be named `main`
    - Avoid infinite loops or anything requiring user interaction

## Managing Custom Tools

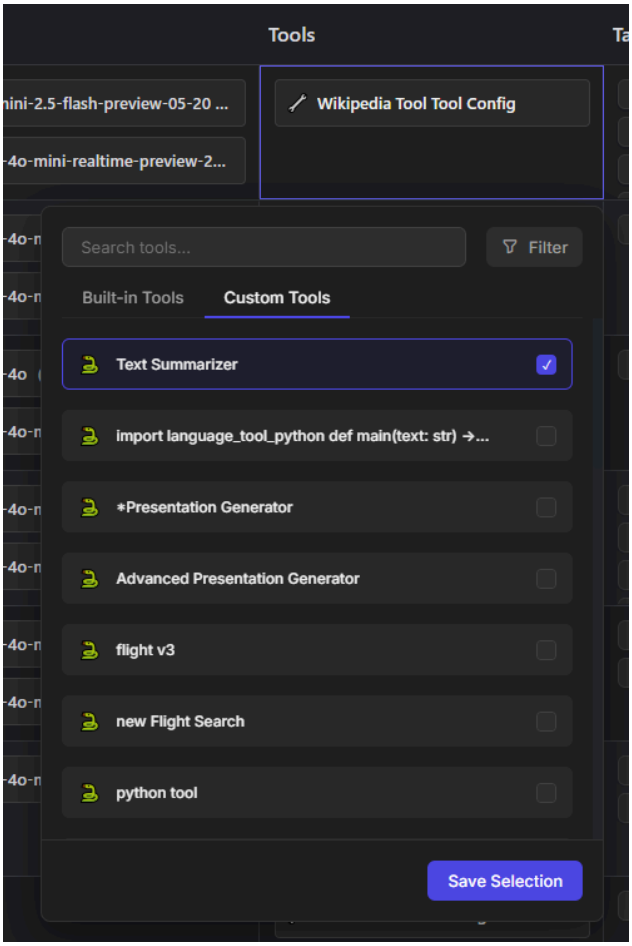
- All created tools appear on the **Custom Tools** tab on the Tools page
- To **edit**, hover over the tool and click **Configure**
- To **delete**, hover and click the **× (cross)** icon




---

## Assigning a Custom Tool to an Agent

1. Go to the **Staff Table**
2. Double-click the cell under the **Tools** column
3. In the popup, switch to the **Custom Tools** tab
4. Select the tool you want to assign from the list



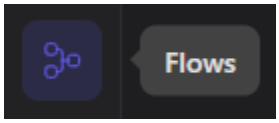

Read how to create and configure agents here: Staff

Flows let you automate how agents and tools interact in your project. Think of it as a visual pipeline where you define *what happens, in what order, and under what conditions*. Every project must be launched through a flow.

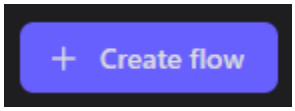
---

## Step 1: Create a New Flow

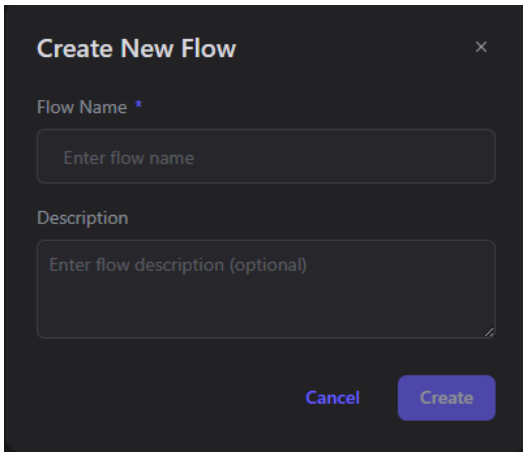
1. Go to the **Flows** page




2. Click the ➕ **Create Flow** button (top right corner)




3. A small window will open – enter:
   - **Flow Name** (required)
   - **Description** (optional)




4. Click **Create**

You'll now be taken to the flow's canvas – a blank space where you'll build the logic of your flow.

---

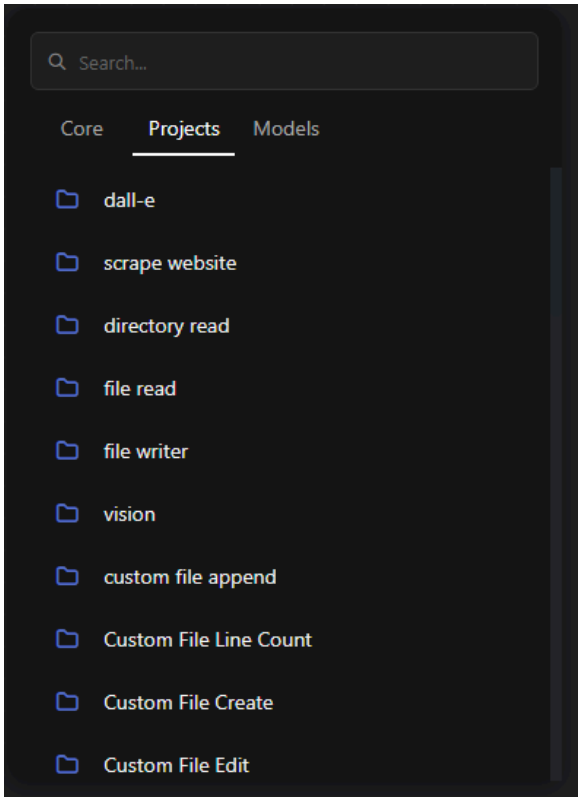
## Step 2: Understand the flow canvas

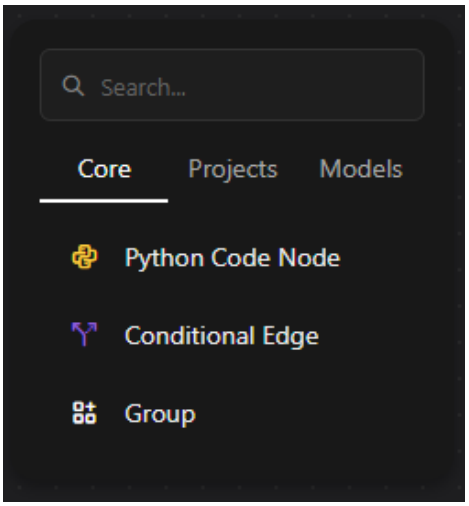
Each flow starts with a **Start Node** – it's added automatically and is always the entry point.

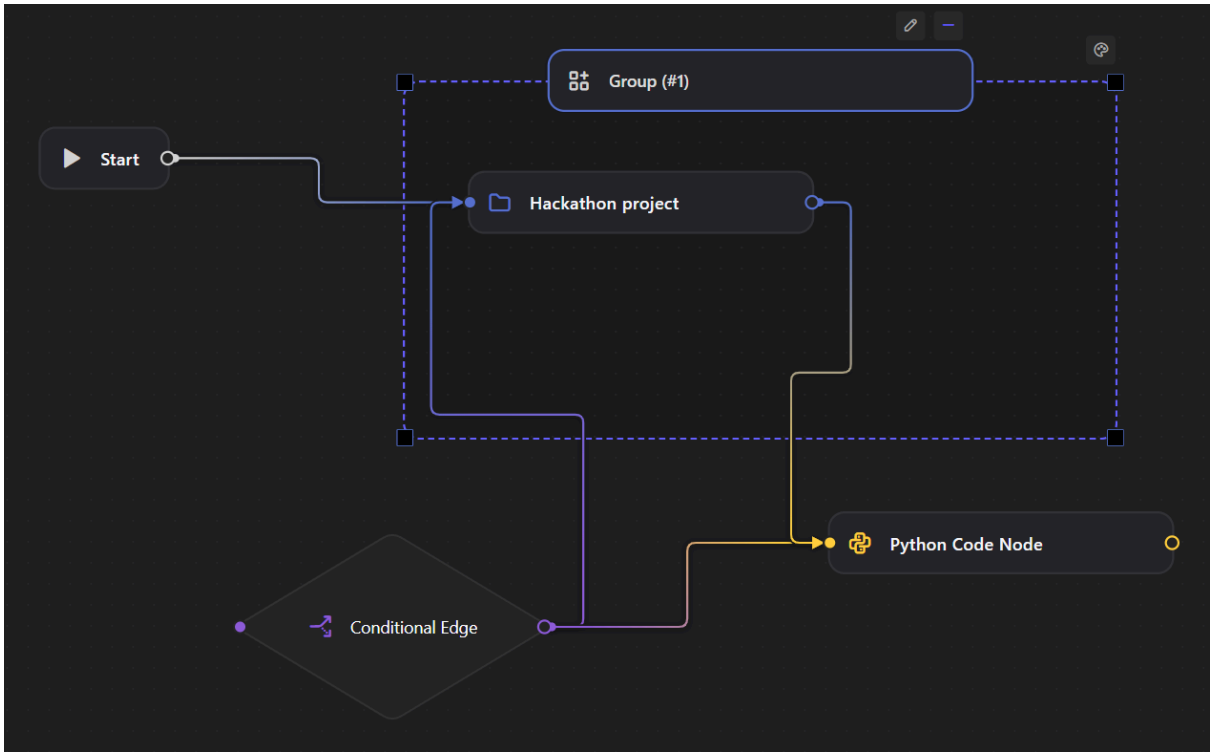
From there, you can add and connect different types of nodes:

- **Python Node** – run Python code
- **Project Node** – run previously created projects
- **Conditional Edge** – set conditions (like if/else branches)
- **Group** – organize your nodes visually

You can find out more about nodes and how to manage them here: Nodes






---

## Step 3: Add nodes to the canvas

1. **Right-click** anywhere on the canvas
2. A popup will appear with node types
3. Navigate the tabs and click the type you want to add

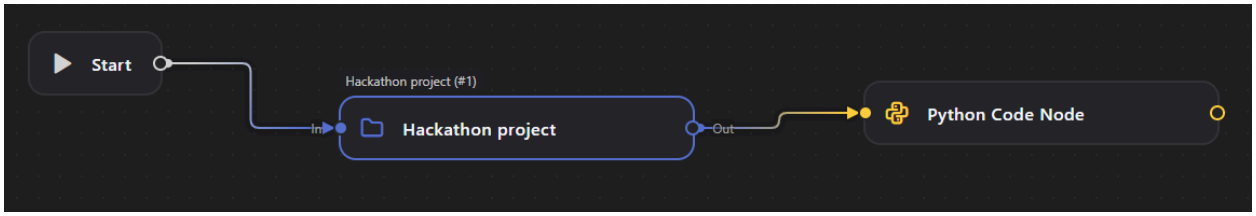
The node will be placed on the canvas. You can **drag to reposition it** as needed.

For better navigation you can click on the magnifying glass icon, then you will see the list of all nodes present on your canvas. Click on any node and you will be redirected to it.

---

## Step 4: Connect the nodes (Define the Flow)

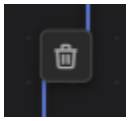
1. **Click and drag** from one node's output port to another node's input
2. Arrows will appear between them
3. This defines the **execution order** – from Start to Finish




⚠️ ***Important***: Your flow will run *exactly* in the order of these connections.

To **delete a connection**:

Click on the arrow and press **Delete** or use the **trash icon**



To delete a node: click it, then use the delete key or the trash icon.

---

## Step 5: Configure each node

Each node has **its own settings**:

- Click on a node to open its configuration panel
- You can:
    - Set **variables** (e.g., pass data from one node to another)
    - Rename the node (each node must have a **unique name**)

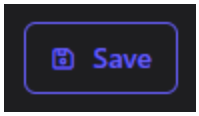
We explain **variables** separately – but they let nodes exchange data and results. Here's the page dedicated to variables

---

## Step 6: Save and run the Flow

Before running, you **must save** your flow:

1. Click **Save** (top right corner)




2. Then click **Run**

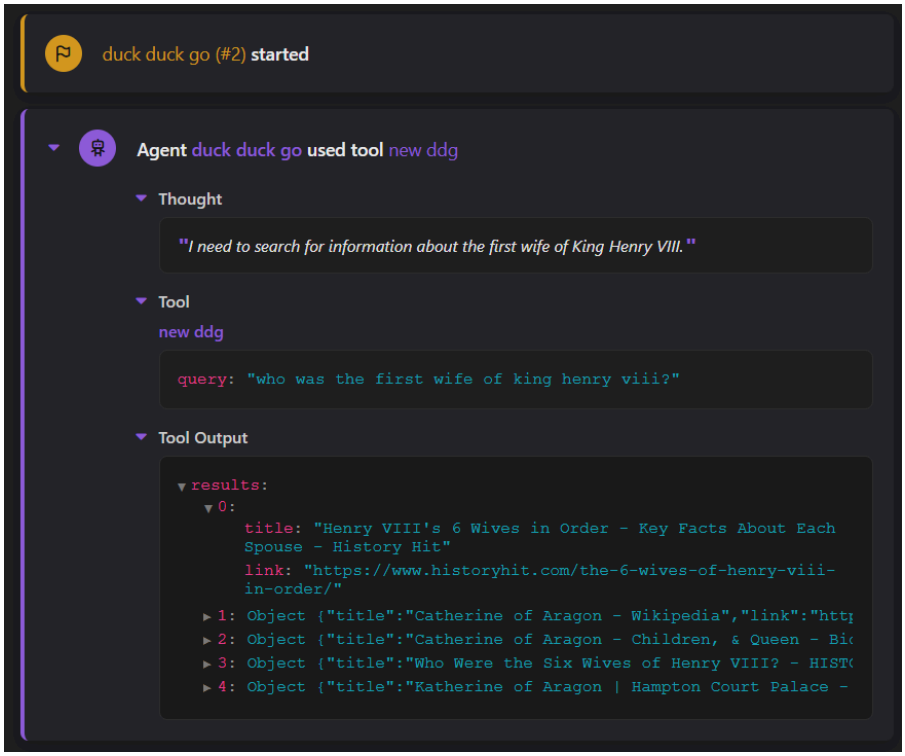
You will be redirected to the **Run Session Page**, where you can watch your flow execute live.

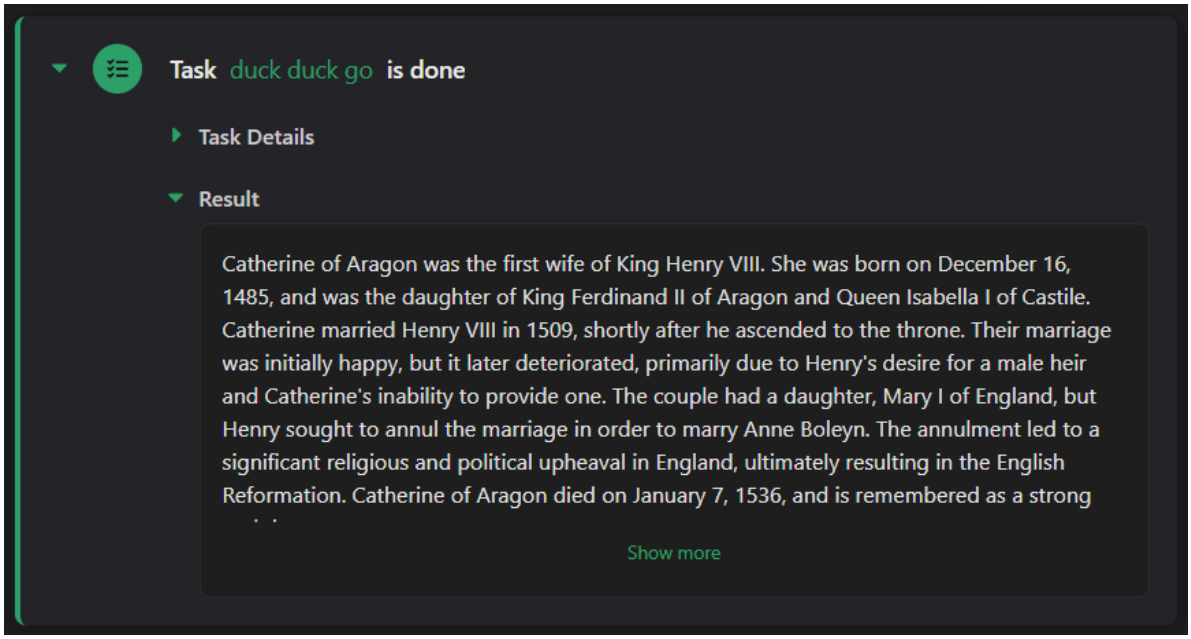
---

## Step 7: What happens during execution?

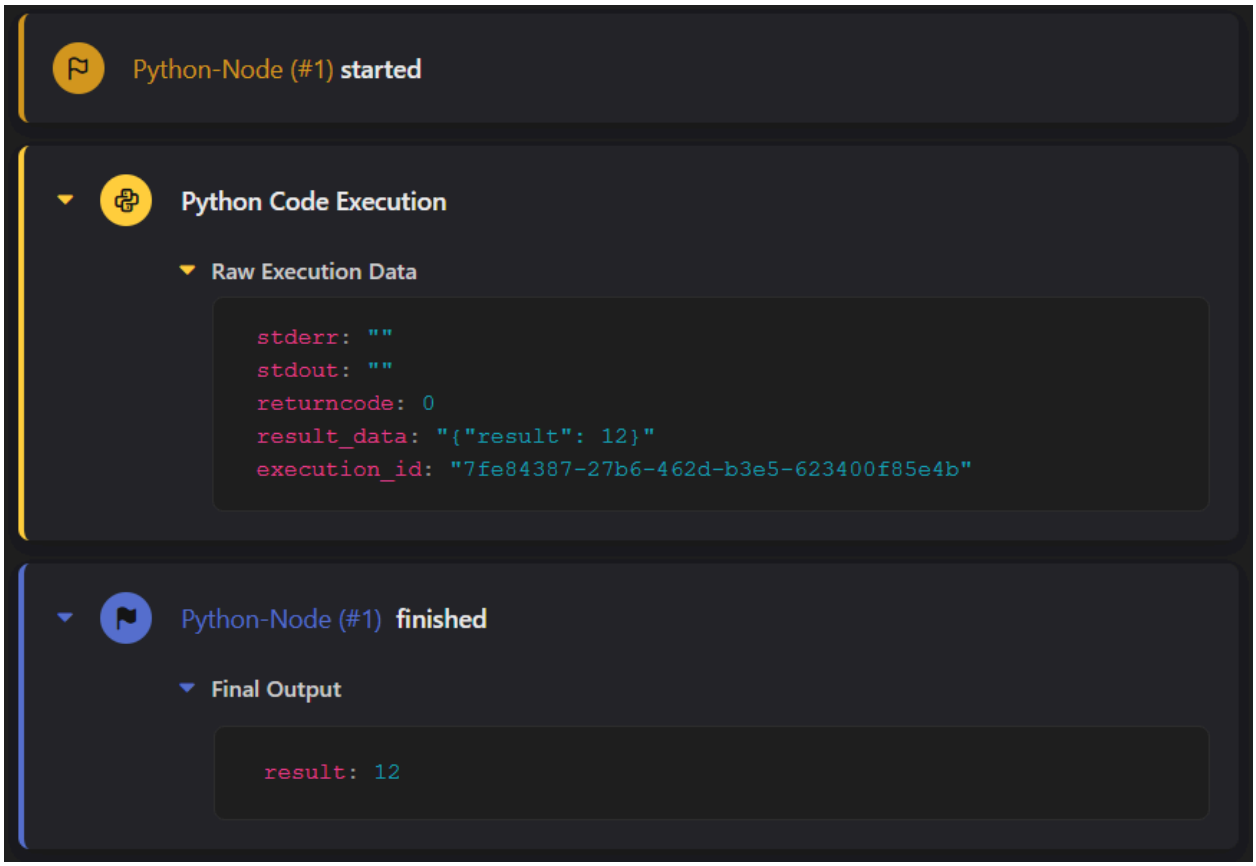
As your flow runs:

- You'll see nodes activate one by one, **in the defined order**
- For each **Project Node** (agent task):
    - You'll see:What the agent was asked to do
        - Their thought process (reasoning)
        - If a tool was used, you'll see:
            - The **query**
            - The **tool output**
        - Final result or decision


duck duck go (#2) started
Agent duck duck go used tool new ddg
Thought
"I need to search for information about the first wife of King Henry VIII."
Tool
new ddg
query: "who was the first wife of king henry viii?"
Tool Output
results:
0:
title: "Henry VIII's 6 Wives in Order - Key Facts About Each Spouse - History Hit"
link: "https://www.historyhit.com/the-6-wives-of-henry-viii-in-order/"
1: Object {"title":"Catherine of Aragon - Wikipedia","link":"http
2: Object {"title":"Catherine of Aragon - Children, & Queen - Bi
3: Object {"title":"Who Were the Six Wives of Henry VIII? - HIST
4: Object {"title":"Katherine of Aragon | Hampton Court Palace -


Task duck duck go is done
Task Details
Result
Catherine of Aragon was the first wife of King Henry VIII. She was born on December 16, 1485, and was the daughter of King Ferdinand II of Aragon and Queen Isabella I of Castile. Catherine married Henry VIII in 1509, shortly after he ascended to the throne. Their marriage was initially happy, but it later deteriorated, primarily due to Henry's desire for a male heir and Catherine's inability to provide one. The couple had a daughter, Mary I of England, but Henry sought to annul the marriage in order to marry Anne Boleyn. The annulment led to a significant religious and political upheaval in England, ultimately resulting in the English Reformation. Catherine of Aragon died on January 7, 1536, and is remembered as a strong
Show more


duck duck go (#2) finished
Variables
date: "today"
project: Object {"raw":"Catherine of Aragon was the first wife of
Final Output
raw: "Catherine of Aragon was the first wife of King Henry VIII. She was born on December 16, 1485, and was the daughter of King Ferdinand II of Aragon and Queen Isabella I of Castile. Catherine married Henry VIII in 1509, shortly after he ascended to the throne. Their marriage was initially happy, but it later deteriorated, primarily due to Henry's desire for a male heir and Catherine's inability to provide one. The couple had a daughter, Mary I of England, but Henry sought to annul the marriage in order to marry Anne Boleyn. The annulment led to a significant religious and political upheaval in England, ultimately resulting in the English Reformation. Catherine of Aragon died on January 7, 1536, and is remembered as a strong and devout queen."


Python-Node (#1) started
Python Code Execution
Raw Execution Data
stderr: ""
stdout: ""
returncode: 0
result_data: "{"result": 12}"
execution_id: "7fe84387-27b6-462d-b3e5-623400f85e4b"
Python-Node (#1) finished
Final Output
result: 12

---

## Execution Statuses – what do they mean?

At the top of the run screen, you'll see the **flow status**:


- **Completed** – Flow finished successfully
- **Running** – Flow is actively executing
- **Pending** – Flow is waiting to start
- **Waiting for Human Input** – Agent needs a user reply
- **Error** – Something went wrong
- **Expired** – Flow ran too long or was interrupted

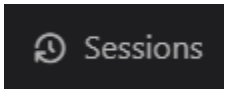
---

## Step 8: Reviewing sessions (past flow runs)

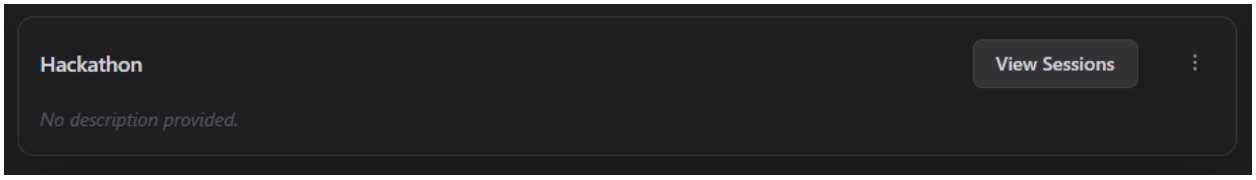
Each time you run a flow, it creates a **session** – a full log of what happened.

You can find past sessions in two places:

1. On the Run Page: click the **Sessions** button (top right)

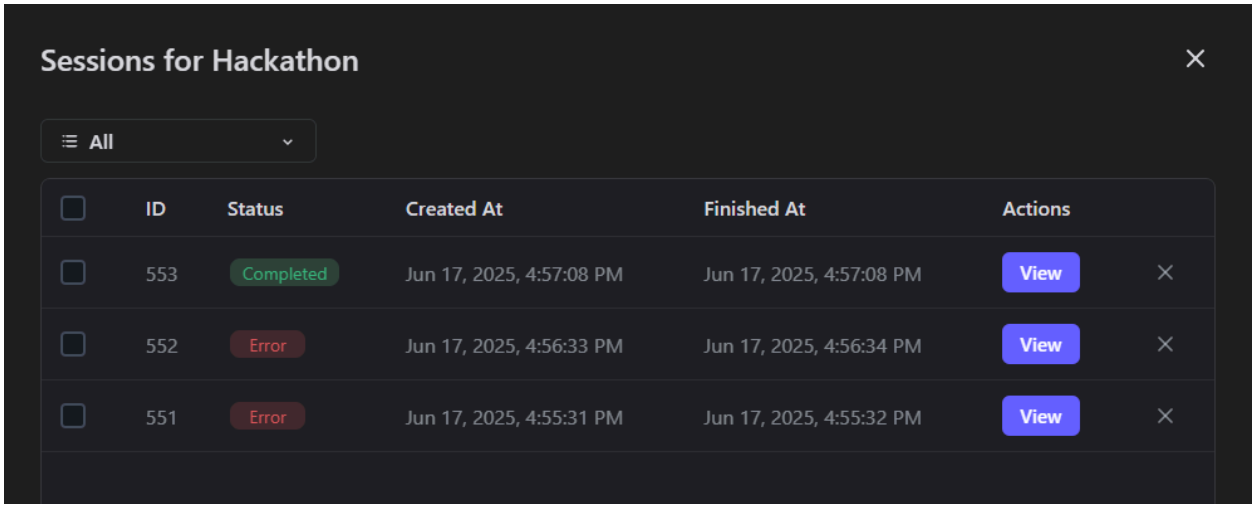



2. On the Flows list: click **View Sessions** next to any flow

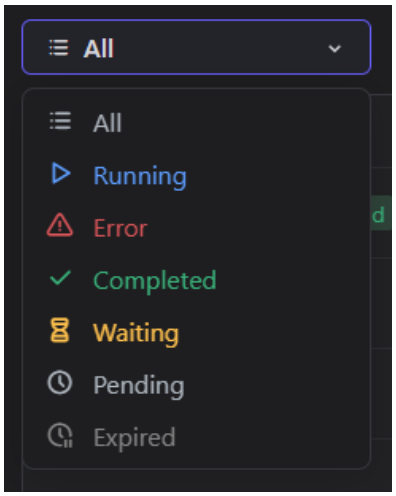



From the sessions panel, you can:

- **Open** any session to review it step-by-step




- **Filter** by status




- **Delete** sessions using the ❌ icon

---

## Important Notes

- You can't run a project *without* a flow
- You **must save** before running – otherwise your latest changes won't be applied
- Use clear and unique names for nodes to stay organized

# Node Types in Flows

Each flow is made up of **nodes**, which are the building blocks of automation. Here's a breakdown of every node type, what it does, and how it can be configured.

## 1. Start Node

- This node is always present by default in every flow.
- It **does not perform any action**, but acts as the entry point of your flow.
- You can pass initial **variables** into it using JSON — these variables will be available to all following nodes.

You can find a detailed explanation about variables here: Variables

---

## 2. Python Node

A Python Node lets you run custom Python code inside your flow. Use it when you need to:

- Transform or filter data
- Do calculations or logic
- Work with external APIs or data
- Prepare values for agents or later steps

---

### Configuration Fields

When you add a Python Node, you'll see these fields:

| Field | 'What it does |
|---|---|
| Name | The name of your Python tool (used in tool lists, not flow logic) |
| Node name | A unique name for this node inside the flow (helps you identify it) |

| | |
|---|---|
| Input map | Defines which flow variables are passed into your Python main() function |
| Output variable path | Where the result will be stored as a flow variable |
| Libraries | Python libraries your code needs. Press Enter/+ after typing each |

You can find a detailed explanation about variables here: Variables

---

**How the main() function works**

Your Python code runs inside a main() function. This function receives inputs based on what you put in the Input map.

*Example 1 — With input map:*

Input map:

```JSON
{
  "x": "variables.price",
  "y": "variables.quantity"
}
```

Python code:

```Python
def main(x, y):
    total = x * y
```

```
    return {"result": total}
```

This will multiply the values of variables.price and variables.quantity, and return the result.

To save the result, you must fill in Output variable path:

```
None
variables.total_price
```

➡ You can use variables.total_price in later nodes or agents.

---

*Example 2 — Without input map:*

If you don’t use any variables or inputs, your code could look like this:

```
Python
def main():
    return {"result": 42}
```

In this case, the node always returns 42 — not dynamic or reusable.

---

**Output variable path**

This tells the system where to store your result from the Python code.

If your main() returns:

```
Python
return {"result": total}
```

And your Output variable path is:

```
None
variables.my_result
```

Then variables.my_result will contain the output and can be used by other nodes.

---

*Summary*

- Use Input map to pass values into main() from your flow.
- Your main() must return a dictionary like {"result": value}.
- Use Output variable path to save the result into a variable.
- If you don't use the input map, your code can still run — but values will be hardcoded.
- Always press Save after making changes to the node!

---

## 3. Project Node

A **Project Node** lets you run any **existing project** inside a flow. It triggers a project you've already built in the system.

### Configuration Fields

When you add a Project Node, you'll see the following fields:

| Field | What it does |
|---|---|
| **Node name** | A unique name for this node inside your flow |
| **Input map** | Pass values from flow variables into the project |
| **Output variable path** | Save the final result of the project into a flow variable |

### How does it work?

When the flow reaches a **Project Node**, it:

1. Loads the selected **project**
2. If provided, uses the values from the **Input map** to populate variables inside the project
3. Executes the project step-by-step
4. Can save the final **output** into the variable you define in **Output variable path**

*Example*

Let's say your project expects a variable called city.

Input map:

```JSON
{

  "city": "variables.destination_city"

}
```

If variables.destination_city is "Paris", then the project will run using city = "Paris".

If the result of the project is a recommendation list, and you write:

```None
variables.recommendations
```

in the **Output variable path**, then the output will be saved and can be used in later nodes.

You can find a detailed explanation about variables here: Variables

---

## 4. Conditional Edge

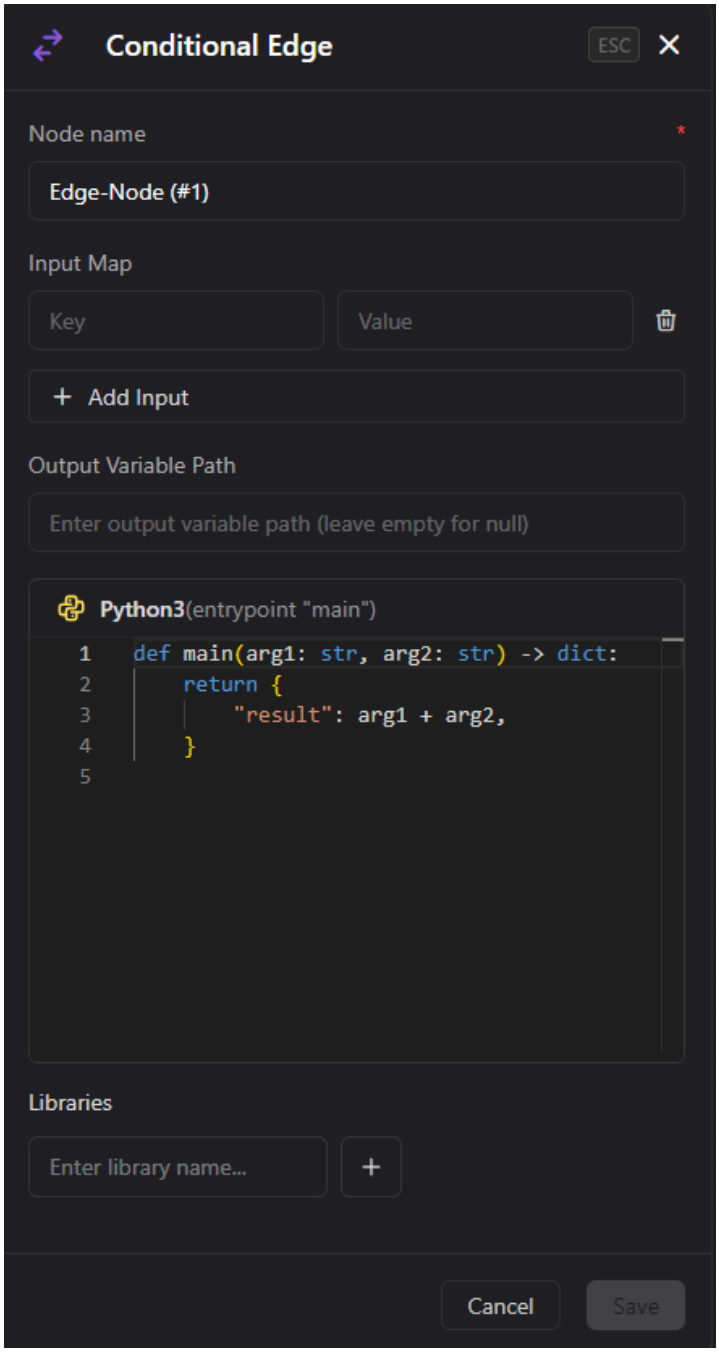
A **Conditional Edge** is a special type of node that helps your flow decide **which path to follow next**, based on logic written in Python.

It's like an "if/else" statement for your flow.

### Configuration Fields

When you click on a Conditional Edge node, you'll see these configuration fields:

| Field | Description |
|---|---|
| **Node Name** | The unique name of this node inside the flow |
| **Input Map** | Here you define which input variables the node should receive (e.g., {"input_value": "my_var"}) |
| **Output Variable Path** | If you want to save a result (like a decision), specify the variable name here |
| **Libraries** | List any extra Python libraries your code needs (remember to press Enter or click + after each) |
| **Python Code Window** | This is where you write the logic that evaluates which condition to follow |




## How does it work?

1. In the **Python code box**, you write logic like:

```python
if variables.input_value > 10:
```

```
    return "path_a"
else:
    return "path_b"
```

2. Based on what you return (like `"path_a"` or `"path_b"`), the flow will **follow the edge** with a matching label.

3. You can connect **two or more outputs** from the conditional edge to other nodes — each with a **label** matching the values returned from the code.

---

**How to set up the connections**

- After writing your conditional logic, **drag an edge from the output port** to the next node.
- You'll be prompted to **enter a condition label** (like `"path_a"`, `"path_b"`, `"default"` etc.).
- The flow will follow the edge that **matches the returned label** from your Python code.

*Example*

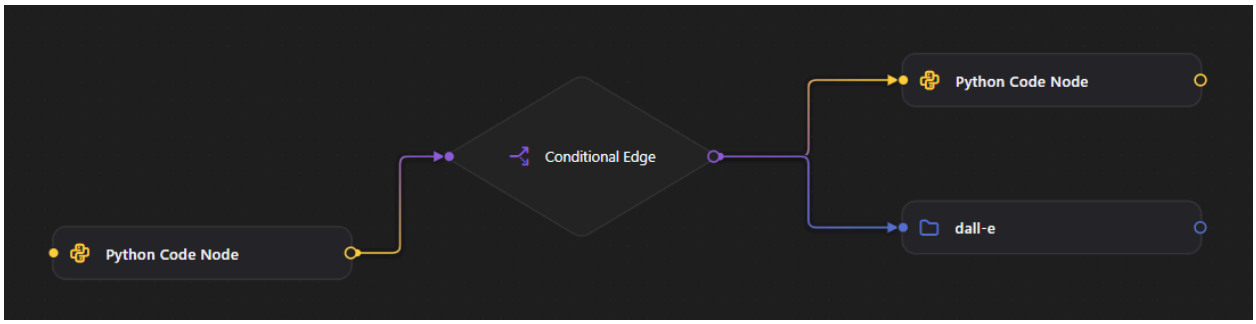
If your condition code is:

```python
if variables.status == "approved":
    return "go"
else:
    return "stop"
```

You should connect the conditional edge to two nodes:

- One with the label `"go"`
- One with the label `"stop"`

The flow will automatically route to the correct one based on the logic.




---

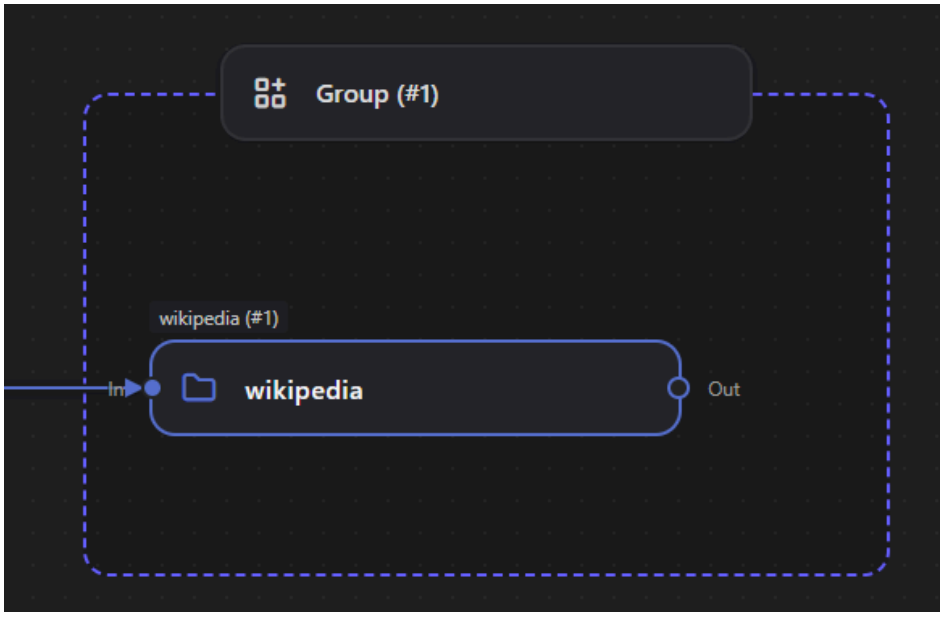
## 5. Group

A **Group** is a visual container inside your flow that helps organize nodes together. It's useful for keeping related logic in one place, especially in large flows.

### How to use Groups

### Add nodes to a Group:

- Just **drag and drop** any node into the group box.
- Once inside, the node becomes part of that group.

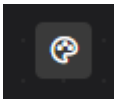


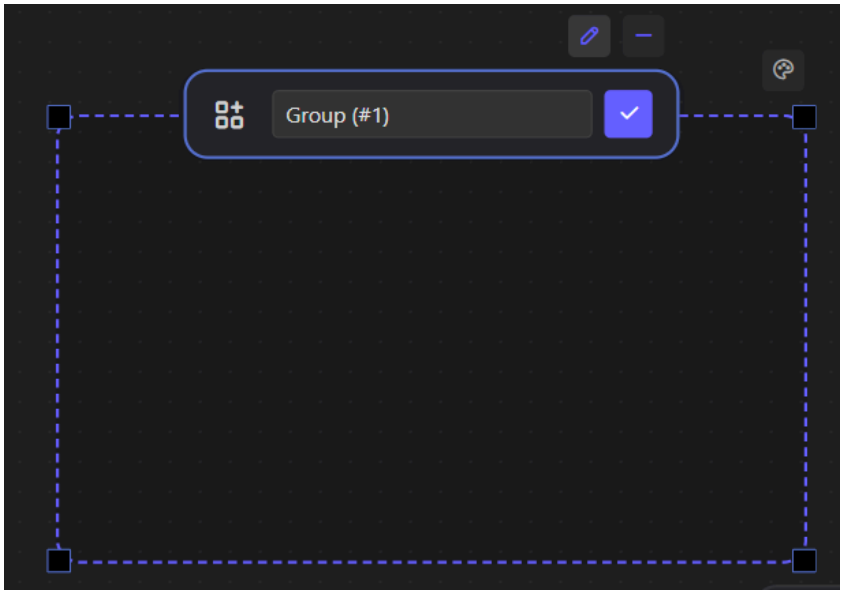

## Remove a node from a Group:

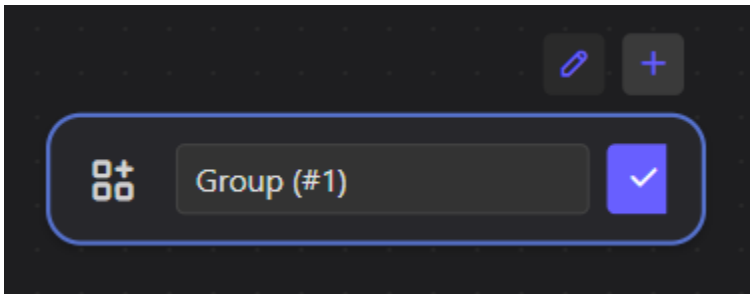
- **Hover over the node**, and you'll see a **special "remove from group" button** appear (usually near a corner). 
- Click that button to take the node out of the group.

## Group Options

You can customize the appearance and behavior of each group:

| Feature | Description |
|---|---|
| **Rename Group** | Click the group name to edit it |
| **Change Color** | Use the color picker to change the background of the group  |
| **Collapse/Expand** | Click the **+ / –** button near the group name to hide or show its contents |




---

**Why use Groups?**

- Keep your flow clean and easy to understand
- Visually separate different logic sections
- Make large flows easier to debug and manage

# What are variables in flows?

**Variables** are a way to **pass data between nodes** in a flow.

They act like named containers that store values (such as text, numbers, or outputs from a node), and they make it possible for different parts of your flow to share and reuse data.

---

# Where and how are variables used?

Variables are used in multiple parts of the flow system:

## 1. Start Node

- This is where the flow begins.
- You can set **initial values for variables** here.
- Example:
  In the Start Node configuration, you might define:

```json
{
  "first_name": "Anna",
  "city": "Paris"
}
```

These variables can now be used by all other nodes.

More about nodes here: Nodes

---

## 2. Input Map

Every node (Python node, project node, conditional edge) has an **Input Map** field.

This is where you tell the node **which variables to use** as its inputs.

**Example:**

If your node expects a variable named `first_name`, your Input Map should look like this:

```JSON
{
  "name": "first_name"
}
```

Inside your Python code or project, you can now access the variable as:

```Python
variables.name
```

More about nodes here: Nodes

---

## 3. Output Variable Path

If your node **returns a result**, you can store that result in a new variable using this field.

Example:

- You write Python code that returns a value like:

```Python
return {"result": greeting}
```

- In **Output Variable Path**, you type:

```None
variables.my_result
```

Now, a new variable called `my_result` will be available for the next nodes.

More about nodes here: Nodes

---

## Passing variables between nodes

Once a variable is defined or updated in one node, it can be passed along and used in the **Input Map** of any other node in the flow. This creates a **chain of data** between nodes.

More about nodes here: Nodes

---

## Using variables in projects via `{}`

When working with **Project Nodes**, you can pass variables into the agent's task prompt or project template using **curly braces** `{}`.

*Example:*

If you defined a variable `destination = "Rome"` in the Start Node, you can write this in your project task:

> "Find cheap flights to **{destination}**"

The system will replace `{destination}` with `"Rome"` automatically when the flow runs.

⚠️ This only works if:

- The variable was already defined in a previous node (like the Start Node)
- It is available under `variables.destination` in the flow

More on how to create and launch project here: Projects, Flows

---

## Variables in python code (main)

Inside a Python Node, variables are always accessed like this:

```python
input_value = variables.my_variable
```

You should never hard-code values inside your `main()` function unless it's for testing or temporary usage.

*Correct way:*

```python
def main():
    name = variables.user_name
    return {"result": f"Hello, {name}!"}
```

*Temporary (not recommended for final setup):*

```python
def main():
    name = "John"  # Not using variables
    return {"result": f"Hello, {name}!"}
```

---

## What happens if a variable is missing?

If you reference a variable that doesn't exist:

- The node might crash or return an error
- The flow could stop working at that point

Always make sure that:

- The variable is either defined in the **Start Node**
- Or it was created earlier in the flow using another node's **Output Variable Path**

---

## Summary: Key Rules for Variables

| Rule | Explanation |
| --- | --- |
| Variables must be passed using the `variables.` prefix | `variables.city`, `variables.user_age`, etc. |
| Input Map connects flow variables to node inputs | Tells the node what values to use |
| Output Variable Path stores a node's result | Makes it available to future nodes |
| In Project Node prompts, use `{}` to insert variables | Example: `"Hello, {first_name}!"` |
| In Python code, always access values through `variables.` | Not by hardcoding! |
| Variables are global inside the flow | Once created, they can be used by any future node |

# What is Human Input?

**Human Input** is a way to let a person (the user) manually respond during a project flow. When Human Input is active, the system **pauses the flow** and waits for the user to reply — just like a chat.

It's useful when:

- The task needs confirmation or clarification from the user
- You want to ask questions and get open-ended responses
- A human decision is required before continuing

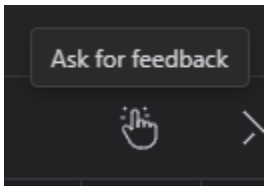
---

## How to enable Human Input

Human Input can only be activated **per task**.

**Here's how:**

1. Go to the **Project Page**
2. Find the **row** with the task you want to enable Human Input for
3. Check the box next to **"Human Input"** in that row
   Once checked, this task will now wait for user input when triggered in a flow




📌 **This is the only way to activate Human Input.**
If the checkbox isn't enabled, the task will run without asking the user.

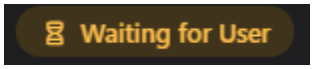
More on how to create and launch a project here: Projects, Flows

---

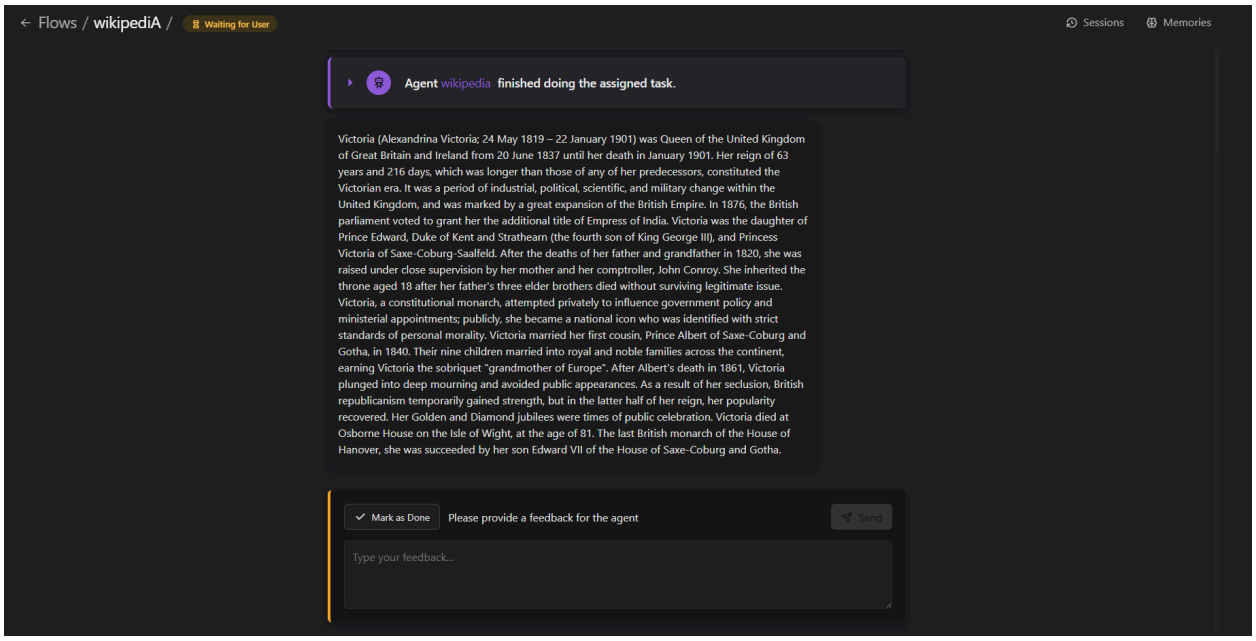
## What happens during execution?

When a flow reaches a task with **Human Input** enabled:

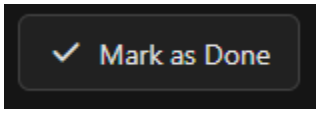
1. The flow will **pause** and session's status will change to "Waining for user"

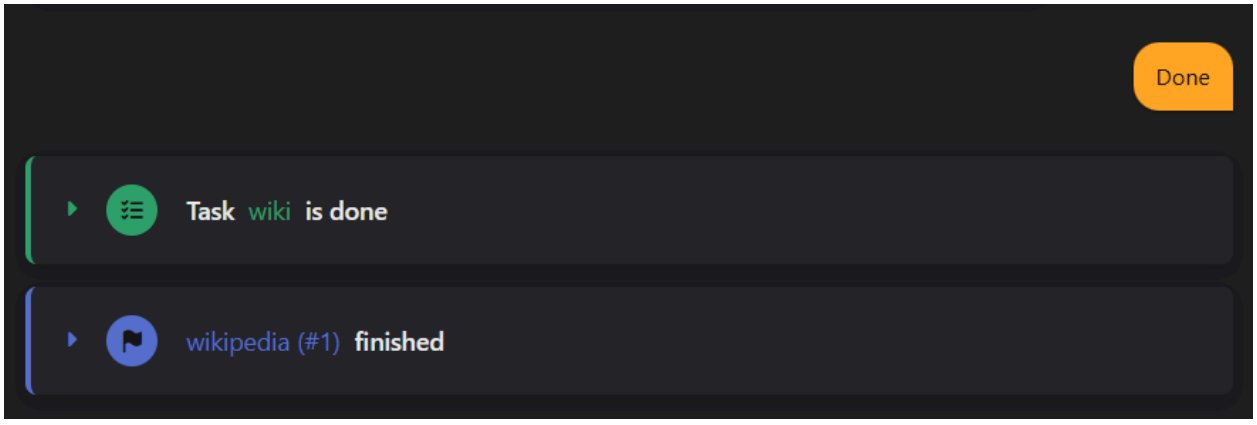



2. A **chat window** appears asking the user to respond

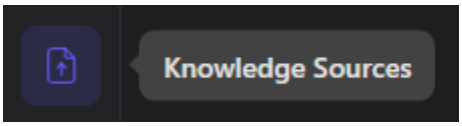



3. The user can **type replies as many times as needed**
4. To continue the flow, the user must click **"Mark as Done"**. Without clicking that, the flow will stay paused



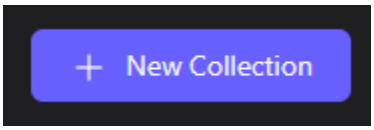
Agents can be enhanced with external documents and data through **Knowledge Sources**, allowing them to reference structured or unstructured information **before generating responses**. This is especially helpful for giving agents access to domain-specific materials, manuals, reports, or other context that they weren't trained on.




## Create a Knowledge Collection

To add a knowledge source:

1. Go to the **Knowledge Sources** tab.
2. Click **+ New Collection** (top right).

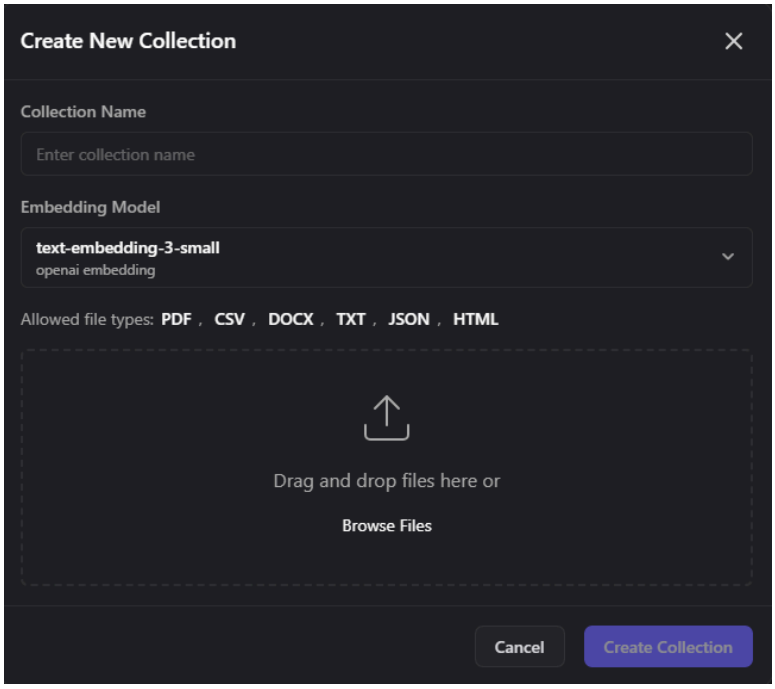


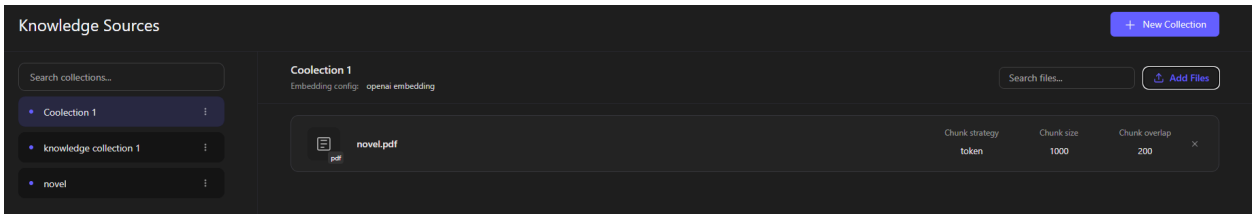

### Required fields:

- **Collection Name** (mandatory): Give your collection a clear, descriptive title.
- **Embedding Model** (mandatory): Choose the embedding model used to vectorize the content (this determines how the system "understands" and retrieves relevant chunks of your documents).
- Choose strategy, chunk size and chunk overlap parameters

Find out how to create an embedding model here: LLMs

- **Upload Files**:
  Select one or more files to include in the collection.
  **Allowed file types**: PDF, CSV, DOCX, TXT, JSON, HTML.




---

## File Settings: Chunking strategy

Once a file is uploaded, you'll configure how it's broken down ("chunked") for vector search.

### Parameters:

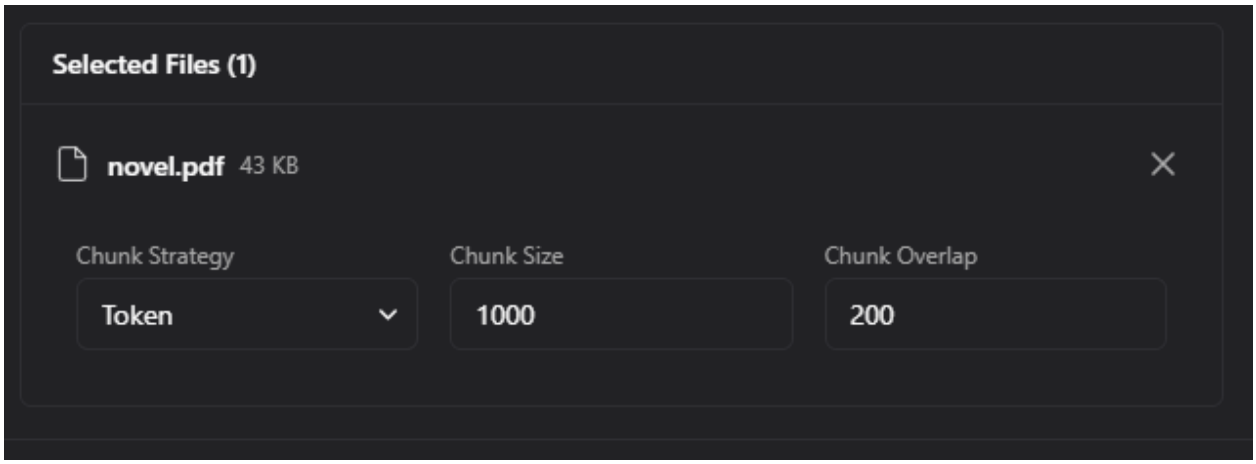
1. **Chunk Strategy** – Choose how to split the content:
    - **Token** – Splits based on number of tokens (language model units).
    - **Character** – Splits based on character count.
    - **Markdown** – Parses and splits using markdown structure (headers, lists).
    - **JSON** – Parses JSON objects.
    - **HTML** – Parses structured HTML content.

2. **Chunk Size** – The size of each chunk (in tokens, characters, etc., depending on the strategy). Affects how much content the model can reference at once.

**Tip**: Larger sizes = more context, but slower search.

3. **Chunk Overlap** – Number of tokens/chars that repeat between chunks. Prevents splitting mid-sentence or mid-paragraph.

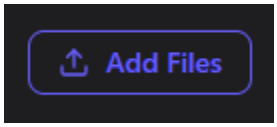
   ⚠️ Must be smaller than chunk size.




---

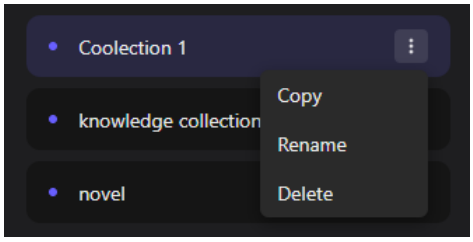
## Managing files in a collection

- To **add more files** later:
  Click **Add Files** in the top-right corner of a collection.




- To **remove a file**:
  Click the **X** icon next to the file name.

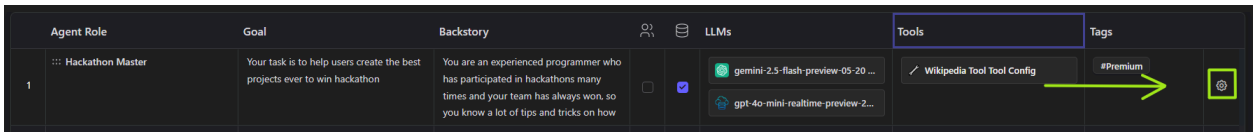
- To **rename or delete a collection**:
  Click the **three dots** beside the collection title.



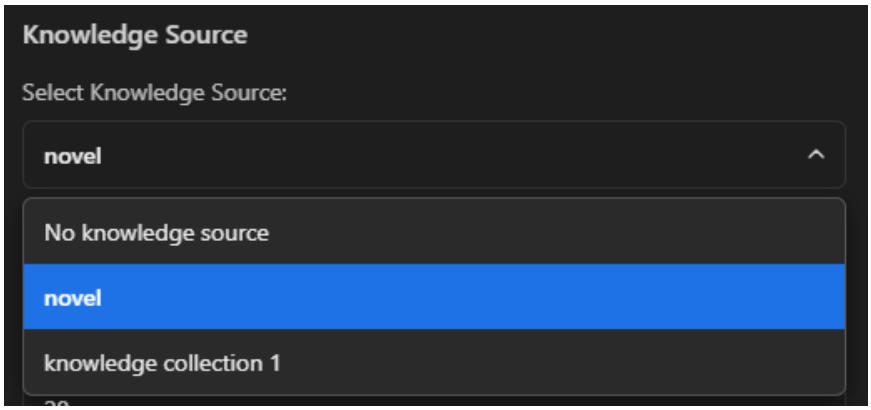
## Assigning a knowledge source to an agent

Once your collection is ready:

1. Open the **Agents** table.
2. Click the **gear icon (⚙️)** next to the agent you want to configure.




3. Scroll to the **Knowledge Source** section.
4. Under **Select Knowledge Source**, pick your desired collection from the dropdown.




5. **Save** changes.

Now the agent will reference that collection during execution.

Read how to create agents here: Agents

## Why Use Knowledge Sources?

- Give agents access to internal data, guides, research, or instructions.
- Improve task accuracy and domain-specific reasoning.
- Avoid having to "teach" the same information in every prompt.

Memory allows agents or projects to **remember information during a session**. This can help them:

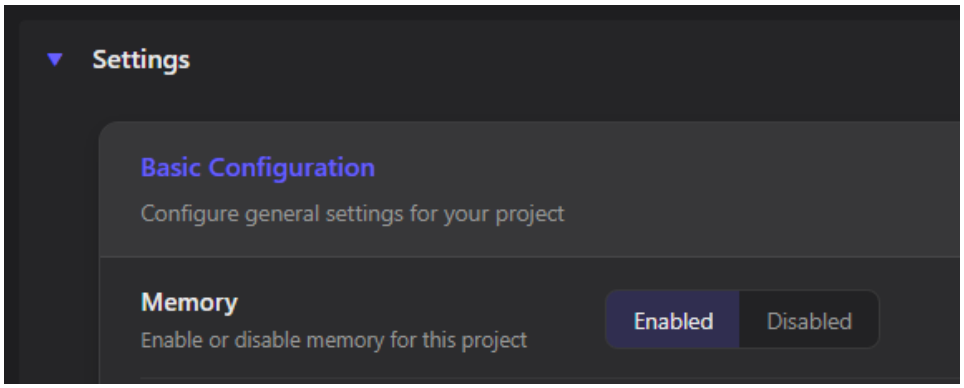
- Refer back to previous results
- Remember user input
- Track important facts as the flow progresses

Currently, only **project memory** is functional (agent memory is not yet active).

---

## How to enable project memory

1. Go to a specific **Project page**
2. Open **Settings**
3. Toggle the switch to **enable memory**




Once enabled, memory will automatically work during each **flow run** of that project.

⚠️ Important:

- Memory is **session-based** — meaning it resets with every new run.
- If you start a new session (even with the same flow), the memory is **cleared**, and a new one begins.

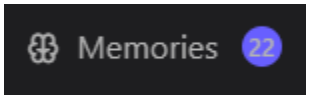
More on how to create and launch a project here: Projects, Flows

---

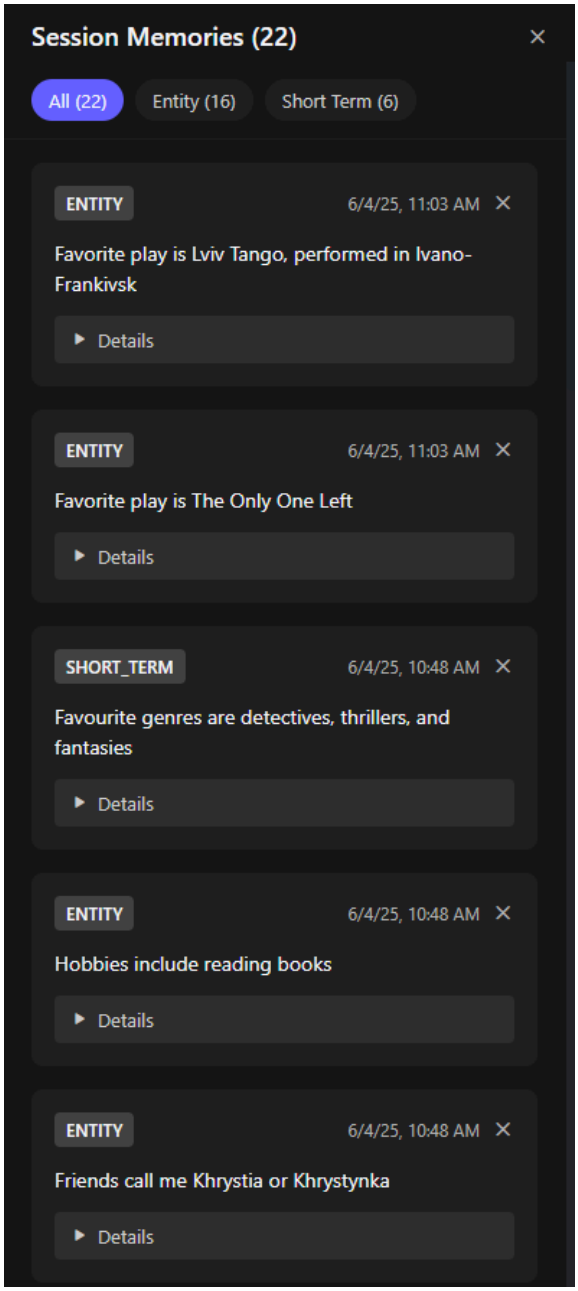
## How to view memory during a flow run

When your project is running in a flow (on the **Run Session Page**):

1. Look at the **top-right corner** of the screen
2. Click the **“Memories” button**




This opens a **sidebar panel** showing the memory contents for that session.



---

## Types of memory

There are **four types of memory**, each serving a different purpose:

### 1. Short-Term Memory

- Automatically stores **temporary thoughts or facts** gathered by the agent during reasoning.
- Useful for immediate next steps.
- Example: intermediate steps in solving a problem.

### 2. Long-Term Memory

- Automatically stores **important knowledge** the agent decides is worth keeping during the session.
- Typically higher-level conclusions, goals, or facts.

### 3. Entity Memory

- Tracks **structured information** like:
    - Names
    - Dates
    - Locations
    - Other extracted data from text

- Example: if the user says “My name is Olivia”, the system may store:

```
None
name: Olivia
```

### 4. User Memory

- Only created when a user **types something manually** using a Human Input node.
- Stores whatever the user enters, so it can be referenced later.

More about Human input here: Human Input

---

## Interacting with Memory

From the memory sidebar, users can:

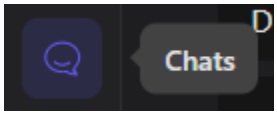
- **View each memory entry**
- **Delete** any specific memory item
- **Filter** by memory type (or view all combined)

---

## Summary

| Feature | Description |
|---|---|
| Project Memory | Active, session-based, enables agents to remember info |
| Agent Memory | Not available yet |
| Activation | Enable in project settings |
| Access | Available via “Memory” button during a flow run |
| View/Edit | You can read or delete memory entries from the sidebar |
| Types | Short-Term, Long-Term, Entity, User Memory |

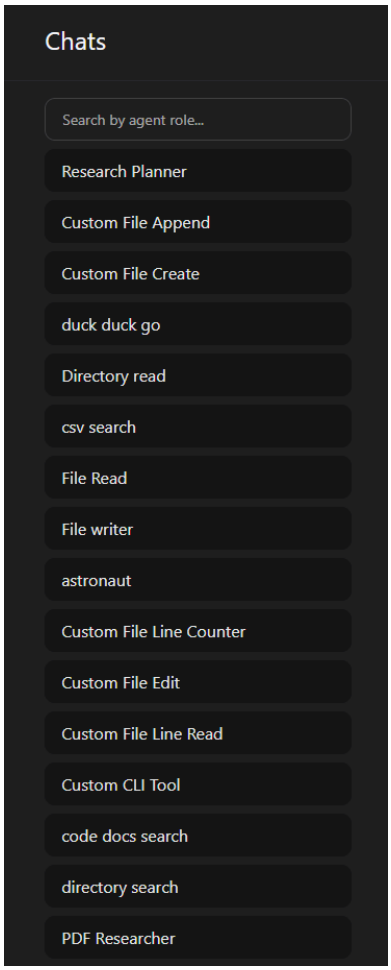
The **Realtime Chat** page allows you to talk to your agents **via voice or text** in real time. This feature is useful for testing agents interactively or having a natural conversation with them.




## Agents Displayed

On this page, you will see **all agents you've created** in the system – the same ones listed in your **Staff Table**. (Read how to create and configure agents here: Staff )

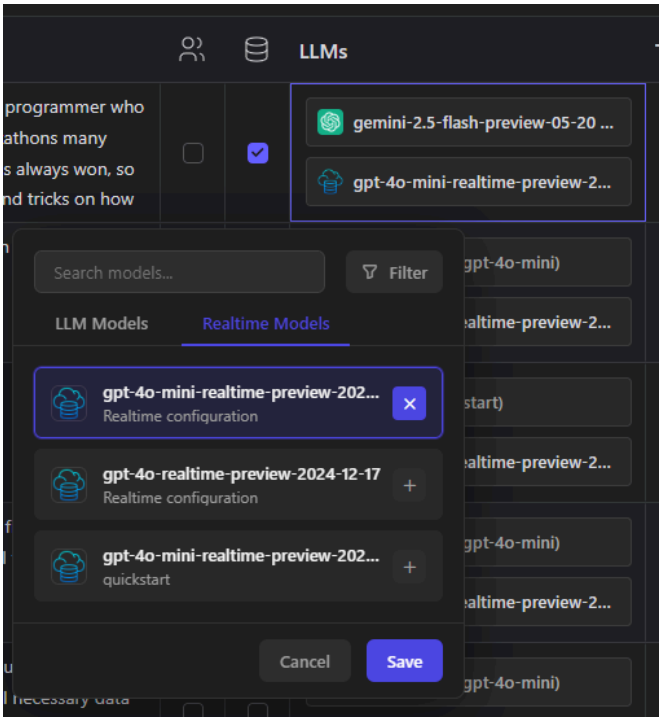
⚠️ **Only agents with a configured Realtime LLM model can respond in this chat.**



## Requirements to Enable Voice Chat

Before an agent can respond in Realtime:

1. **You must assign a Realtime LLM model**
    - Go to the **Staff Table**
    - Double-click the `LLMs` cell of your agent
    - Open the **Realtime Models** tab in the popup
    - Click the ➕ button to assign a Realtime model (only one can be active)

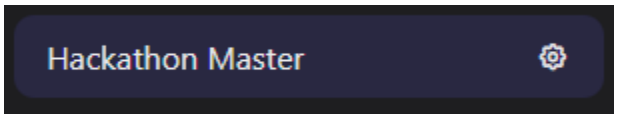


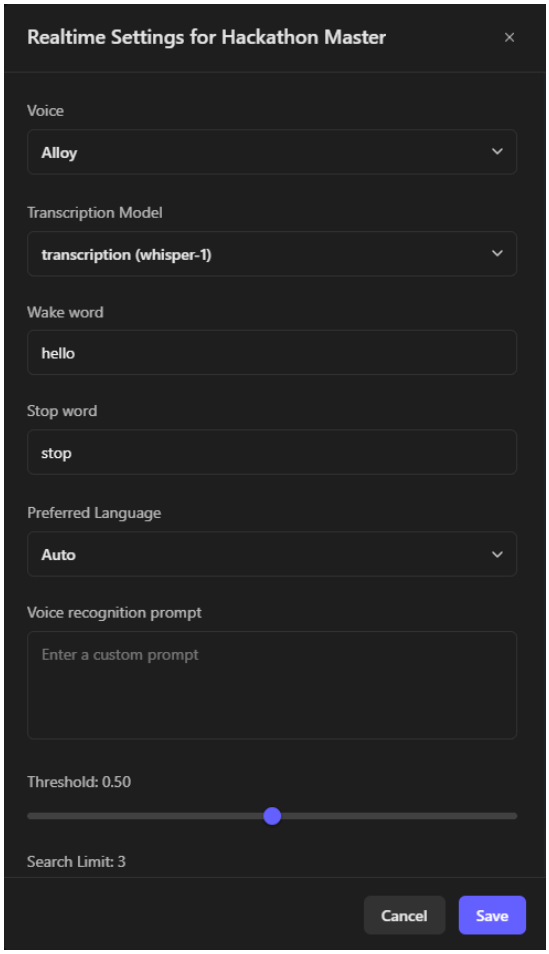

A detailed instructions on how to set up a realtime LLM is here: LLMs

2. **Agent must be configured in the Realtime Chat page**
   Next to the agent's name, click the **gear icon** to open their **Realtime Settings**.

---

## Realtime Agent Settings

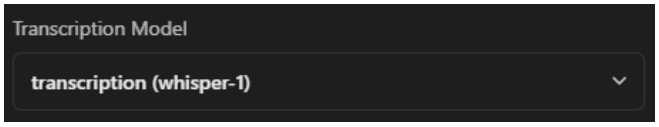





In the Realtime Settings modal, you'll see:

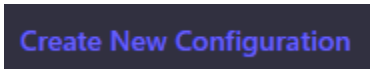
- **Voice**: Choose a voice profile (cosmetic only – it doesn't affect functionality)


- **Transcription Model** (*required*):
  Used to convert your speech to text. Without it, the agent won't hear or respond.

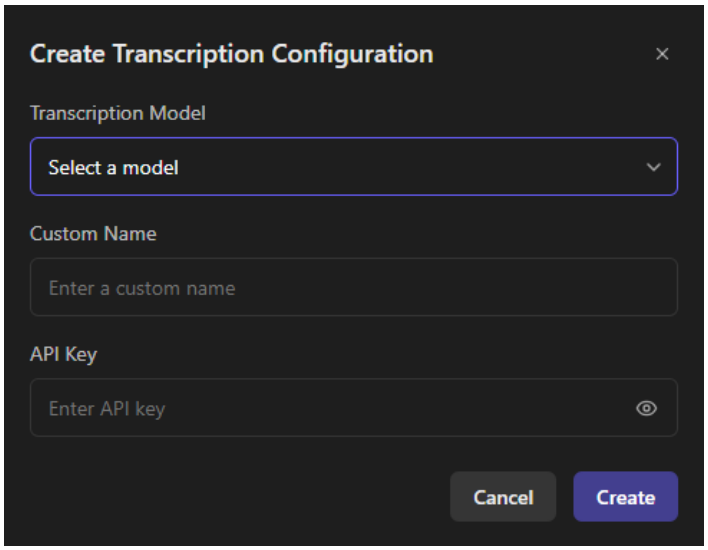



### If no Transcription Model exists:

- Click the dropdown → **Create New Configuration**




- Fill out the modal form:
  - **Transcription Model**: Select one from the list (dropdown)
  - **Custom Name**: Give it a name
  - **API Key**: Add your service key if required



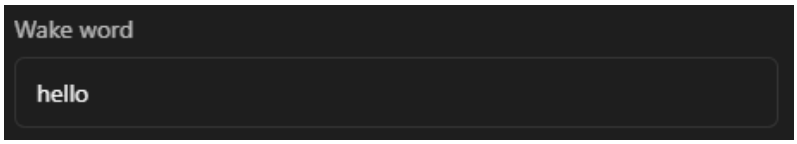
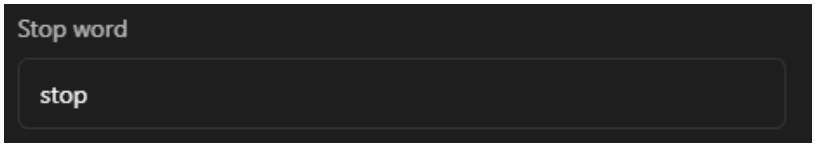
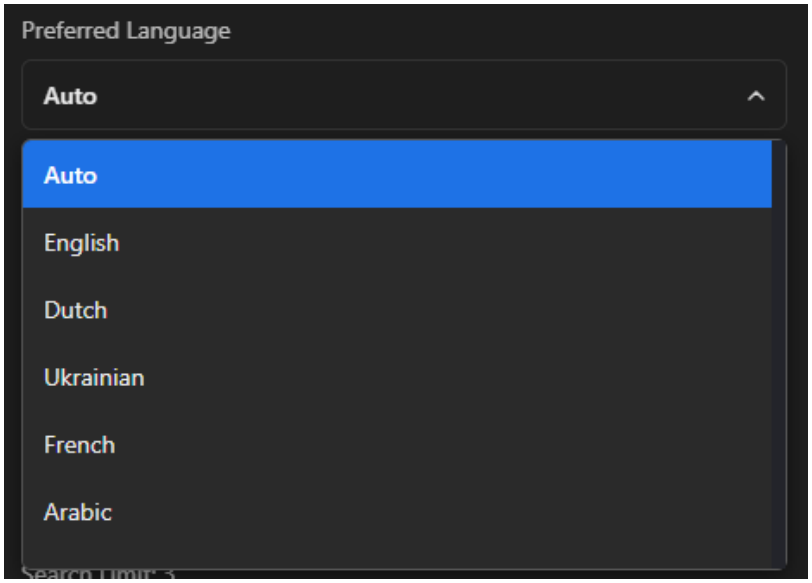

- Once the required fields are filled, the **Create** button becomes active.
- Click it to save and select your new configuration.

---

## Additional Realtime Settings (Optional)

These options are shown in the agent's Realtime Settings window:

| Field | Description |
|---|---|

| | |
|---|---|
| **Wake Word** | A word that activates the agent (e.g., "Hey assistant", “Hello”, Agent’s name)<br>  |
| **Stop Word** | A word that stops the agent from listening<br>  |
| **Preferred Language** | Language used for voice recognition<br>  |
| **Voice Recognition Prompt** | A hint to improve transcription accuracy (e.g., “You are listening to technical commands”). Also good practice to mention agent wake word here |
| **Threshold (0–1)** | Threshold for searching by knowledge. Lower = more accurate knowledge will be extracted. Default: 0.6 |
| **Search Limit (0–1000)** | Limits how many knowledge chunks will be extracted. Default: 3 |

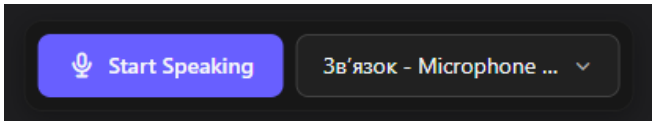
*If you’re unsure what these fields do, ask your developer team before changing them.*

---

## How to Start Talking to Your Agent

Once the LLM and transcription settings are configured:

1. **Allow microphone access** in your browser when prompted
2. Select the agent you want to talk to
3. Click **Start Speaking**

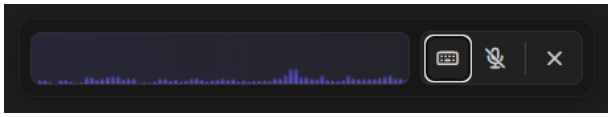


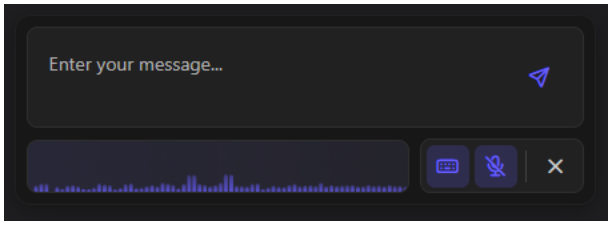

---

## Chat Interface Controls

During the chat session, you'll see:

| Icon | Function |
|---|---|
| **Microphone (when white)** | Tap to speak – the system listens and responds |
| **Microphone (when purple)** | Click to mute yourself; the agent will **not** listen |
| **Keyboard** | Click to type a message instead of speaking |
| **X (Close Chat)** | Ends the conversation – **all messages will be lost** |






⚠️ **Note**: Realtime chats are **not saved**. Once the session is closed, all messages are gone.