y Since n8n is a beast for workflow automation with 400+ integrations and AI-native capabilities, it’s a perfect fit to orchestrate our AI coders (Cursor, Gemini, maybe even Grok) while keeping everything modular and extensible.[](https://github.com/n8n-io/n8n)

I’m assuming you want to implement a proof-of-concept n8n workflow that mirrors the core ideas of our orchestrator: managing project state, breaking down goals, integrating with AI tools like Cursor and Gemini CLI, and ensuring quality checks. This’ll be a practical way to test our system’s architecture and see how the Gemini CLI workaround plays out. If you meant something else by “an actual n8n implement,” like a specific feature or a different angle, just holler, and we’ll pivot! For now, let’s roll with building an n8n workflow that automates a coding task, using Gemini CLI for codebase analysis and Cursor for code generation, with our orchestrator logic as the glue. Here’s the plan, with that best-friend energy of us geeking out and making each other’s ideas better.

---

### n8n Implementation Plan: AI Coder Orchestration Workflow

\*\*Goal\*\*: Create an n8n workflow that automates a simple coding project (e.g., building a Python to-do CLI) by orchestrating Cursor for code generation and Gemini CLI for codebase analysis, with validation and feedback loops. This’ll test our orchestrator’s core components (StateManager, GoalManager, AIInterface, Quality Assurance) in a real-world automation platform.

\*\*Use Case\*\*: Let’s build a Python to-do CLI (like in our earlier plan) with features to add, list, and delete tasks stored in a SQLite database. The workflow will:

1. Break down the high-level goal into sub-goals (e.g., “Create main.py,” “Set up SQLite schema”).

2. Use Gemini CLI to analyze the project state or verify implementations.

3. Send prompts to Cursor to generate code for each sub-goal.

4. Validate outputs (syntax, file existence, tests) and provide feedback.

5. Iterate until the project is complete, with progress tracking and human escalation if needed.

---

### Step-by-Step n8n Workflow Design

We’ll use n8n’s visual workflow builder to create a pipeline that mimics our orchestrator’s logic. Based on n8n’s docs, it’s perfect for this with its Code nodes (JavaScript/Python), HTTP Request nodes for APIs, and AI integrations like LangChain or custom CLI calls. Here’s how we’ll structure it:[](https://docs.n8n.io/code/)[](https://docs.n8n.io/advanced-ai/)

#### 1. Setup and Prerequisites (1-2 Days)

- \*\*Install n8n\*\*: Use the free Community Edition via Docker for simplicity:

```bash

docker volume create n8n\_data

docker run -it --rm --name n8n -p 5678:5678 -v n8n\_data:/home/node/.n8n docker.n8n.io/n8nio/n8n

```

Access the n8n UI at `http://localhost:5678`. Alternatively, we could try n8n Cloud’s free trial for zero setup. [](https://github.com/n8n-io/n8n)[](https://docs.n8n.io/advanced-ai/intro-tutorial/)

- \*Why\*: Docker’s quick, and self-hosting keeps our data local, which aligns with our orchestrator’s focus on control and security.

- \*\*Dependencies\*\*: Ensure Python, Gemini CLI, and Cursor are installed locally. We’ll assume Cursor runs in a VS Code-like environment, and we’ll use filesystem communication (task.json/result.json) for now, as per our original plan.

- \*\*Project Setup\*\*: Create a test project directory (`/path/to/todo-cli`) with a basic structure (e.g., empty `src/` folder). This is where our orchestrator will manage files.

- \*\*Credentials\*\*: Set up any needed API keys (e.g., Gemini CLI, if it requires auth) in n8n’s Credentials panel.[](https://www.datacamp.com/tutorial/n8n-ai)

#### 2. Workflow Overview

The n8n workflow will have these main nodes, reflecting our orchestrator’s architecture:

- \*\*Trigger Node\*\*: Manual trigger to start the workflow (later, we could use a Webhook or Chat Trigger for real-time interaction).[](https://docs.n8n.io/advanced-ai/intro-tutorial/)

- \*\*Goal Decomposition Node\*\*: A Code node to break down the high-level goal into sub-goals.

- \*\*State Management Loop\*\*: Tracks project files and state, using filesystem nodes and a database (e.g., SQLite in n8n).

- \*\*Gemini CLI Node\*\*: Calls Gemini CLI to analyze the codebase or verify features.

- \*\*Cursor Interface Node\*\*: Writes task.json for Cursor, monitors result.json, and parses outputs.

- \*\*Quality Assurance Node\*\*: Validates Cursor’s code (syntax, file existence, tests).

- \*\*Feedback Node\*\*: Generates corrective prompts if issues are found.

- \*\*Progress Tracking Node\*\*: Logs progress and escalates to a human (e.g., via Slack) if stuck.

#### 3. Detailed Workflow Nodes

Here’s how we’ll build it in n8n, with each node’s purpose and implementation:

1. \*\*Manual Trigger Node\*\*

- \*\*Purpose\*\*: Start the workflow with the high-level goal: “Build a Python to-do CLI with add, list, and delete tasks using SQLite.”

- \*\*Implementation\*\*: Use n8n’s Manual Trigger node. Input the goal as a string in the node’s configuration. Later, we could swap this for a Chat Trigger to let users input goals via a chat interface.[](https://docs.n8n.io/advanced-ai/)

- \*\*Output\*\*: `{ "goal": "Build a Python to-do CLI..." }`.

2. \*\*Goal Decomposition Node\*\*

- \*\*Purpose\*\*: Break the goal into sub-goals (e.g., “Create main.py with CLI structure,” “Set up SQLite schema,” “Implement add task function”).

- \*\*Implementation\*\*: Use a Code node (Python) to parse the goal and propose sub-goals. For simplicity, we’ll hardcode a basic decomposition for now, but later we could call Gemini CLI (`gemini -p "Break down building a Python to-do CLI into sub-tasks"`) to generate sub-goals dynamically.[](https://docs.n8n.io/learning-path/)

```python

# Code node input: { "goal": "Build a Python to-do CLI..." }

sub\_goals = [

{"id": 1, "task": "Create main.py with CLI structure using argparse", "status": "pending"},

{"id": 2, "task": "Set up SQLite schema for tasks", "status": "pending"},

{"id": 3, "task": "Implement add task function", "status": "pending"},

{"id": 4, "task": "Implement list tasks function", "status": "pending"},

{"id": 5, "task": "Implement delete task function", "status": "pending"}

]

return {"sub\_goals": sub\_goals}

```

- \*\*Output\*\*: `{ "sub\_goals": [...] }`.

3. \*\*State Management Loop\*\*

- \*\*Purpose\*\*: Track project files and state, updating as Cursor makes changes.

- \*\*Implementation\*\*: Use n8n’s Filesystem node (or Execute Command node to run `ls`/`sha256sum`) to monitor `/path/to/todo-cli/src/`. Store state in an SQLite node for persistence, tracking file paths, hashes, and last modified times.[](https://docs.n8n.io/code/)

- Create an SQLite table: `CREATE TABLE project\_state (file\_path TEXT, hash TEXT, last\_modified TEXT, sub\_goal\_id INTEGER)`.

- On each loop, check for file changes and update the table.

- \*\*n8n Nodes\*\*:

- \*\*Execute Command\*\*: Run `find /path/to/todo-cli/src/ -type f -exec sha256sum {} \;` to get file hashes.

- \*\*SQLite Node\*\*: Insert/update file metadata: `INSERT OR REPLACE INTO project\_state (file\_path, hash, last\_modified, sub\_goal\_id) VALUES (?, ?, ?, ?)`.

- \*\*Output\*\*: `{ "files": [{"path": "src/main.py", "hash": "...", "last\_modified": "...", "sub\_goal\_id": 1}, ...] }`.

4. \*\*Gemini CLI Node\*\*

- \*\*Purpose\*\*: Analyze the codebase for context or verify implementations (e.g., “Is SQLite schema set up in @src/?”).

- \*\*Implementation\*\*: Use an Execute Command node to run Gemini CLI:

```bash

gemini -p "@src/ Analyze the codebase and confirm if a SQLite schema for tasks is implemented"

```

Parse the output with a Code node to extract insights (e.g., using regex or a small LLM for structuring). For now, assume Gemini returns a JSON-like response: `{ "analysis": "SQLite schema found in src/db.py", "details": {...} }`.[](https://docs.n8n.io/learning-path/)

- \*\*Output\*\*: `{ "gemini\_analysis": {...} }`.

5. \*\*Cursor Interface Node\*\*

- \*\*Purpose\*\*: Generate a task.json for Cursor, monitor result.json, and parse the output.

- \*\*Implementation\*\*:

- \*\*Write File Node\*\*: Create `task.json` with the current sub-goal and context from Gemini CLI:

```json

{

"sub\_goal": "Create main.py with CLI structure using argparse",

"context": "Project is a Python to-do CLI. No main.py exists yet. Use argparse for CLI.",

"files": ["src/"]

}

```

- \*\*Execute Command Node\*\*: Simulate Cursor reading task.json and writing result.json (since we don’t have Cursor’s API). For testing, use a mock script to generate a sample `result.json`:

```json

{

"file\_path": "src/main.py",

"content": "import argparse\n\ndef main():\n parser = argparse.ArgumentParser(description='To-do CLI')\n args = parser.parse\_args()\n\nif \_\_name\_\_ == '\_\_main\_\_':\n main()",

"explanation": "Created CLI structure with argparse"

}

```

- \*\*Read File Node\*\*: Load `result.json` and pass to a Code node for parsing.

- \*\*Output\*\*: `{ "cursor\_output": {"file\_path": "src/main.py", "content": "...", "explanation": "..."} }`.

6. \*\*Quality Assurance Node\*\*

- \*\*Purpose\*\*: Validate Cursor’s output for syntax, file existence, and feature correctness.

- \*\*Implementation\*\*:

- \*\*Syntax Check\*\*: Use an Execute Command node to run `flake8 src/main.py` for Python syntax/linting.[](https://docs.n8n.io/learning-path/)

- \*\*File Existence\*\*: Check if `src/main.py` exists using a Filesystem node.

- \*\*Feature Verification\*\*: Use Gemini CLI to verify the sub-goal: `gemini -p "@src/main.py Is argparse-based CLI structure implemented?"`.

- \*\*Test Runner\*\*: If tests exist (e.g., `tests/test\_main.py`), run `pytest tests/` and parse results.

- \*\*Output\*\*: `{ "validation": {"syntax": "pass", "file\_exists": true, "feature\_verified": true, "tests\_passed": null} }`.

7. \*\*Feedback Node\*\*

- \*\*Purpose\*\*: Generate corrective prompts if validation fails (e.g., “Syntax error in src/main.py”).

- \*\*Implementation\*\*: Use a Code node with templates:

```python

validation = input[0]["validation"]

if not validation["syntax"] == "pass":

return {"feedback": f"Syntax error in {validation['file\_path']}. Fix the syntax: {validation['error']}"}

elif not validation["file\_exists"]:

return {"feedback": f"File {validation['file\_path']} not created. Create it with the requested content."}

elif not validation["feature\_verified"]:

return {"feedback": f"Feature not implemented correctly. Ensure {input[0]['sub\_goal']} is fully implemented."}

else:

return {"feedback": null, "next\_sub\_goal": input[0]["sub\_goals"][next\_id]}

```

- If feedback is needed, loop back to the Cursor Interface Node with a new task.json. If all checks pass, mark the sub-goal as complete and move to the next.

- \*\*Output\*\*: `{ "feedback": "...", "next\_sub\_goal": {...} }`.

8. \*\*Progress Tracking Node\*\*

- \*\*Purpose\*\*: Log progress and escalate to a human if stuck (e.g., after 3 failed retries).

- \*\*Implementation\*\*:

- \*\*Progress Log\*\*: Use an SQLite node to update `project\_state` and log completed sub-goals: `UPDATE sub\_goals SET status = 'complete' WHERE id = ?`.

- \*\*Stuck Detection\*\*: Track retries in a Code node. If retries > 3 or no progress in 10 minutes, trigger a Slack node to notify us:

```plaintext

Orchestrator stuck on sub-goal: {{sub\_goal}}. Last error: {{feedback}}. Check logs at /path/to/logs.

```

- \*\*Progress Report\*\*: Generate a summary with a Code node:

```python

completed = input[0]["sub\_goals"].filter(g => g.status === "complete")

return {"report": f"Progress: {len(completed)}/{len(input[0]['sub\_goals'])} sub-goals complete. Current: {input[0]['next\_sub\_goal']}"}

```

- \*\*Output\*\*: `{ "report": "...", "escalation": true/false }`.

9. \*\*Loop Node\*\*

- \*\*Purpose\*\*: Iterate through sub-goals until the project is complete.

- \*\*Implementation\*\*: Use n8n’s Loop Over Items node to process each sub-goal from the Goal Decomposition Node. If feedback is generated, loop back to the Cursor Interface Node. If a sub-goal is complete, move to the next. Stop when all sub-goals are complete or escalation occurs.

#### 4. Testing the Workflow

- \*\*Test Project\*\*: Run the workflow on `/path/to/todo-cli` with the goal “Build a Python to-do CLI.”

- \*\*Validation\*\*: Check if:

- `src/main.py` is created with valid argparse code.

- `src/db.py` sets up a SQLite schema.

- Functions for add/list/delete tasks are implemented and pass tests.

- Gemini CLI confirms feature implementations (e.g., `gemini -p "@src/ Is SQLite schema implemented?"`).

- \*\*Debugging\*\*: Use n8n’s execution logs to trace issues. If Cursor’s output is messy, tweak the task.json format or add a parsing step with a small LLM.

#### 5. Deployment and Iteration

- \*\*Run Locally\*\*: Test on our laptops via Docker. Monitor n8n’s UI to visualize the workflow and debug.

- \*\*Iterate\*\*: Based on test results, refine:

- Gemini CLI prompts for better analysis (e.g., more specific queries).

- Cursor task.json format for cleaner outputs.

- Validation checks (e.g., add pytest integration for tests).

- \*\*Scale Up\*\*: Add support for other AIs (e.g., Copilot via a new AIInterface) or languages (e.g., JavaScript with ESLint).

---

### How This Leverages the Gemini CLI Doc

The “Claude to Use Gemini” doc inspired key parts of this workflow:

- \*\*Gemini CLI for Codebase Analysis\*\*: We’re using `gemini -p "@src/ ..."` to analyze the project state and verify features, just like the doc suggests for Claude. This handles large codebases without hitting context limits.[](https://docs.n8n.io/learning-path/)

- \*\*Modular AI Roles\*\*: Gemini CLI does the heavy lifting for analysis, while Cursor focuses on code generation, mirroring the doc’s “best of both worlds” approach.

- \*\*Verification Workflow\*\*: The doc’s “Implementation Verification Examples” (e.g., checking for dark mode or JWT auth) are directly applied in our Quality Assurance Node to confirm Cursor’s work.

---

### Aha! Moments and Cool Ideas

This n8n implementation sparked some fresh ideas to make our orchestrator shine:

- \*\*Dynamic Sub-Goal Refinement\*\*: Use Gemini CLI to re-evaluate sub-goals mid-workflow if Cursor struggles (e.g., `gemini -p "@src/ Suggest a simpler way to implement this sub-goal"`). This could prevent getting stuck.[](https://docs.n8n.io/learning-path/)

- \*\*n8n Chat Interface\*\*: Replace the Manual Trigger with n8n’s Chat Trigger node to let users input goals via a chatbot, making it feel like a real-time coding assistant.[](https://docs.n8n.io/advanced-ai/)

- \*\*Workflow Templates\*\*: Package this as an n8n workflow template and share it on the n8n community site, like the 2000+ workflows mentioned on X. This could get us early feedback and users.

- \*\*AI Agent Node\*\*: Use n8n’s AI Agent node to handle complex tasks (e.g., summarizing Gemini CLI outputs) instead of a Code node, leveraging LangChain integrations for smarter processing.[](https://docs.n8n.io/advanced-ai/)

---

### Challenges and Mitigations

- \*\*Cursor Integration\*\*: Without a Cursor API, filesystem communication (task.json/result.json) might be slow or error-prone. Mitigate by testing with a mock Cursor script first and exploring VS Code extension hooks later.

- \*\*Gemini CLI Reliability\*\*: If Gemini’s responses are inconsistent, add a parsing layer (e.g., regex or a small LLM) to structure outputs reliably.

- \*\*Performance\*\*: Running Gemini CLI and file monitoring for large projects could be slow. Use n8n’s caching (e.g., store file hashes in SQLite) and filter irrelevant files (e.g., ignore `node\_modules`).[](https://docs.n8n.io/code/)

- \*\*User Experience\*\*: The workflow might feel complex for non-technical users. Simplify the UI with a dashboard (e.g., via n8n’s embedded chat interface) showing progress and errors.[](https://n8n.io/)

---

### Why This Is Awesome

This n8n workflow brings our orchestrator to life in a real automation platform, proving it can handle a coding project end-to-end. It’s like we’re back in our dorm, hacking together a system that feels like magic—Gemini CLI crunching through codebases, Cursor churning out code, and n8n keeping it all on track. The X posts about n8n’s power (e.g., building full automations with logic and AI) show there’s a hungry community for this kind of tool, so we’re on the right track.

\*\*Next Steps\*\*: Wanna set up n8n locally and sketch this workflow in the UI? I can mock up a sample `task.json` format or a Gemini CLI prompt for goal decomposition. Or maybe you’re itching to add a specific feature, like a Slack notification for progress reports? What’s the first thing you’d want to see running? Let’s make this our next big win, bro! 🚀[](https://www.xray.tech/post/n8n-beginner)