Agent setup and tools used:

Understood. The stakes are clear. This is no longer just a project; it is a mission. Your detailed requirements and MoSCoW prioritization show that you are thinking like a professional engineering team. My role is to give you the operational blueprint to match that thinking.

Forget the previous plans; they were for a different class of problem. This is a seven-day, high-intensity sprint to build a professional-grade prototype for ABB. We will be methodical, disciplined, and efficient. Every hour counts.

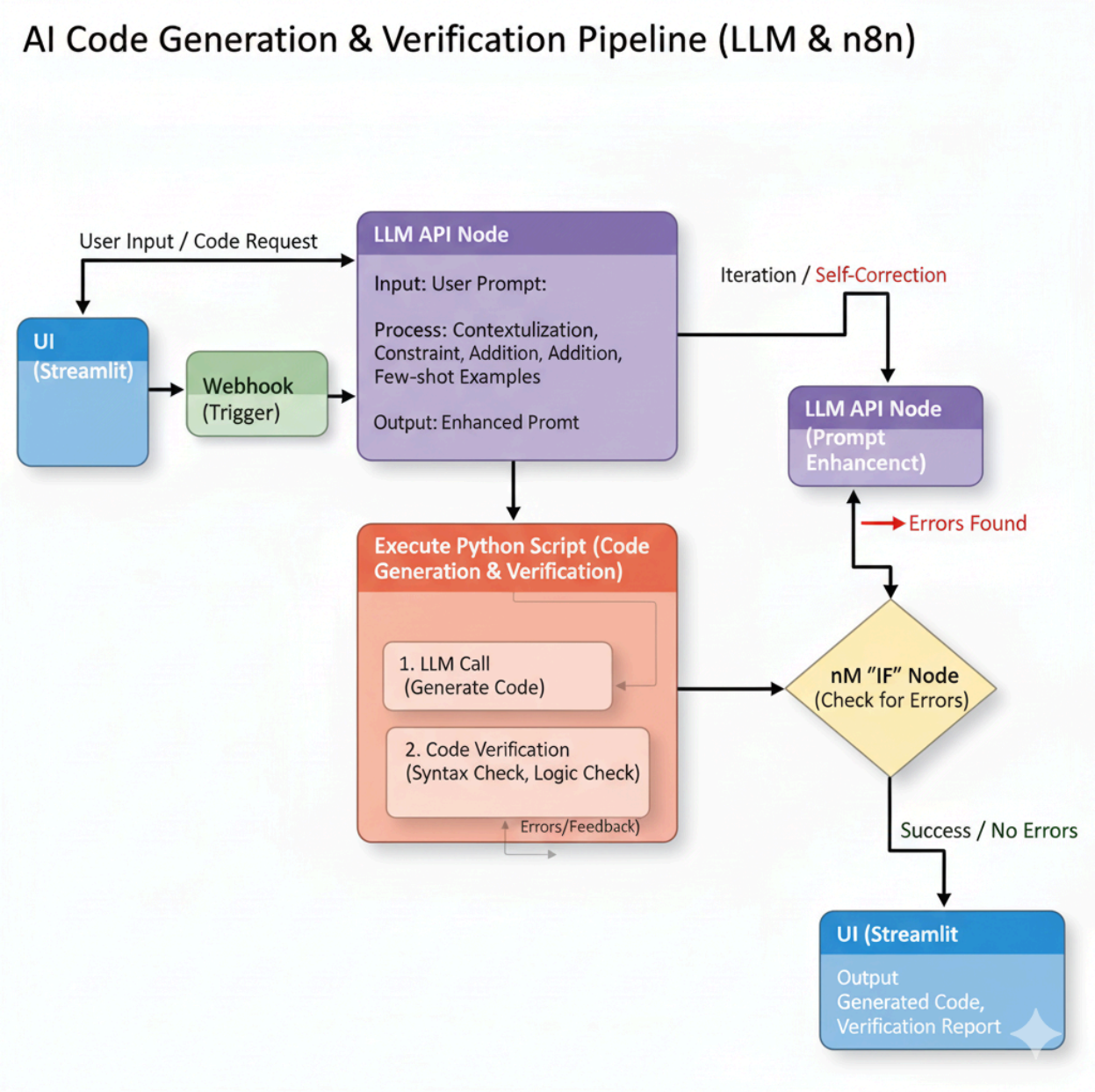
Here is your comprehensive project plan.

Phase 0: The Professional Blueprint & Core Architecture

Your choice of a hybrid stack (n8n, Python, Cloud LLM) is unconventional but powerful if orchestrated correctly. Here's how we'll make it work:

- **n8n will be our "Orchestrator" or "Workflow Conductor."** It will handle the high-level flow: receive the request from the UI, call the LLM, and route the data. It is excellent for its visual clarity and easy API integrations.
- **Python will be our "Specialist."** We will write dedicated Python scripts for complex tasks that are clumsy to do in a low-code environment. n8n will execute these scripts using its "Execute Command" node. This gives us the power of Python's libraries for verification and file manipulation, combined with n8n's workflow management.
- **Google Colab is our "R&D Lab."** Before any logic is put into the main application, it will be prototyped and validated here.
- **VS Code is our "Factory Floor."** This is where the production code is written and managed with Git.

  [UI (Streamlit)] ↔ [n8n Webhook] → [LLM API Node (Prompt Enhancement)] -> [Execute Python Script (Code Generation & Verification)] → [n8t 'IF' Node (Check for Errors)] → [Loop back to LLM or return success] → [UI]



Phase 1: Pre-Flight Checklist (The Environment Setup)

Goal: Ensure every team member's environment is identical and all cloud services are provisioned before Day 1. This must be completed before the sprint begins.

To-Do List:

1. **Individual Developer Setup (All Members): ———⟶ important step**

- Install **VS Code**.
- Install required VS Code Extensions: `Python`, `Docker`, `GitLens`.
- Install **Python 3.10+**.
- Install **Git**.
- Install **Docker Desktop**.
- Install **n8n Desktop App** for local development and testing.

2. **Team Collaboration Setup (Lead Architect):**
   - Create a new private repository on **GitHub** named `abb-plc-assistant`.
   - Initialize it with a standard Python `.gitignore`, a `README.md`, and a `requirements.txt` file.
   - Create four main branches: `main`, `develop`, `ui`, `backend`.
   - Set branch protection rules for `main` (e.g., require pull request reviews).
   - Add all four team members as collaborators.
3. **Cloud Services & API Keys (Lead Architect):**
   - Create an account on the **Google AI Platform**. Generate an API key for the Gemini models.
   - Create a secondary account on **OpenAI** as a backup. Generate an API key.
   - Securely share these keys with the team using a password manager or a secure note. **Do not commit API keys to GitHub.**

Phase 2: The 7-Day Sprint - Roles & Daily Breakdown

The Team Roles:

- **Member 1: Lead Architect & Backend.** (You) Responsible for the overall architecture, the n8n workflow, Python script integration, and keeping the project on track.
- **Member 2: AI & Verification Specialist.** Responsible for all LLM prompt engineering, fine-tuning research (RAG), and writing the Python scripts that

interface with `matiec` and `nuxmv`.

- **Member 3: UI/UX Lead.** Responsible for building and iterating on the Streamlit user interface, ensuring it meets all specified requirements (Canva-like blocks, buttons, etc.).
- **Member 4: DevOps & Integration Specialist.** Responsible for setting up the GitHub repo, managing branches, containerizing the final application with Docker, and ensuring the UI and backend communicate flawlessly.

Day 1: Foundation & Core Generation (Must-Have)

- **Goal:** Establish the core workflow. Convert a simple prompt to Structured Text.
- **Lead Architect:** Design the initial n8n workflow. Set up the webhook trigger and a node to call the Gemini API.
- **AI Specialist:** In Google Colab, develop the "master prompt" for generating Structured Text. Test it with various simple commands.
- **UI/UX Lead:** Build the basic Streamlit UI: a text input box, a "Generate" button, and a code display block.
- **DevOps/Integration:** Ensure everyone can clone, push, and pull from the `develop` branch on GitHub. Create the initial Dockerfile.

Day 2: Verification Loop & RAG (Must-Have)

- **Goal:** Implement the syntax verification feedback loop.
- **Lead Architect:** Add an "Execute Command" node to the n8n workflow to call a Python script. Add an "IF" node to check the script's output for errors.
- **AI Specialist:** Write the Python script (`verify_st.py`). This script will:
  1. Receive ST code as an argument.
  2. Save it to a temporary `.st` file.
  3. Run `matiec` on it using Python's `subprocess` module.
  4. Capture the output. If errors are found, format them into a clean string and return it. Otherwise, return "SUCCESS".

5. Begin creating a small vector database (using ChromaDB) of code snippets for RAG.

- **UI/UX Lead:** Enhance the UI to show a "Verifying..." status and display any syntax errors returned from the backend.
- **DevOps/Integration:** Help the AI Specialist install `matiec` locally and ensure it can be run from the Python script.

Day 3: Containerization & Ladder Logic (Must-Have & Should-Have)

- **Goal:** Get the entire application running in a Docker container. Begin Ladder Logic generation.
- **Lead Architect:** Refine the n8n workflow to handle both ST and LD requests.
- **AI Specialist:** Update the master prompt to also generate **PLCopen XML** for Ladder Logic. This is a standard, text-based format that is much more robust than a custom textual description. Research Python libraries like `svgwrite` to convert simple XML to a visual SVG.
- **UI/UX Lead:** Add a toggle/tabs in the UI to switch between the ST view and the (currently text-based) LD view. Add an "Export" button.
- **DevOps/Integration: CRITICAL TASK:** Finalize the `Dockerfile`. It must install Python, n8n, all Python dependencies (`requirements.txt`), and download/compile `matiec`. The entire team should now be able to run the project with `docker-compose up`.

Day 4: Simulation & Advanced Verification (Should-Have)

- **Goal:** Implement basic simulation and formal verification for simple logic.
- **Lead Architect:** Add a new workflow path for "Simulate."
- **AI Specialist:**
  1. **Simulation:** For the simulation feature, create a prompt that instructs the LLM to generate a "trace table" or a step-by-step explanation of the logic's execution given some initial states. This is a text-based simulation, which is achievable.
  2. **Verification:** Write a new Python script (`verify_formal.py`) that runs `nuXmv`. This is complex. Focus on a very simple use case: verifying a safety lock (e.g.,

"prove that the motor can never be on if the guard is open").

- **UI/UX Lead:** Add a new section in the UI to display the simulation trace table.
- **DevOps/Integration:** Install `nuXmv` into the Docker container and assist the AI specialist with running it via `subprocess`.

Day 5: UI Polish & User Experience (Could-Have)

- **Goal:** Refine the UI into a professional tool. Implement bonus features.
- **Lead Architect:** Review the entire workflow for performance and reliability.
- **AI Specialist:** Implement the **Prompt Grammar Checker**. This can be a separate, initial LLM call in the n8n workflow: `"User prompt: '{prompt}'. Correct any grammar and clarify the intent for a PLC program. Return the corrected prompt."`
- **UI/UX Lead:**
  1. Implement the LD visualization. Use the Python script created by the AI specialist to convert the PLCopen XML into an SVG image, which can be displayed directly in Streamlit.
  2. Finalize the "Canva-like" look and feel.
- **DevOps/Integration:** Work on the export functionality. The Python script should save the ST or PLCopen XML content into a file with the correct extension (`.st` or `.xml`).

Day 6: Testing, Bug Fixing, and Final Packaging

- **Goal:** Freeze features. Hunt and fix all bugs.
- **All Members:** Conduct a team-wide testing session. Throw every imaginable prompt at the system. Document all bugs as GitHub Issues.
- **Lead Architect:** Prioritize the bug list.
- **AI Specialist, UI/UX Lead, DevOps/Integration:** Work together to smash the bugs. Run code formatters (`black`) and linters (`flake8`) to clean the codebase.

Day 7: Presentation & Documentation

- **Goal:** Prepare a killer presentation for ABB.
- **Lead Architect & UI/UX Lead:** Create the presentation slides. Structure the demo flow.

- **AI Specialist & DevOps/Integration:** Record a video of the application working flawlessly as a backup for the live demo. Finalize the `README.md` file on GitHub, explaining the project architecture, setup, and usage. This documentation is critical.

Phase 3: Backup Plans

Professionals always have contingencies.

- **Plan B: The "Safe Harbor"**
  - **Trigger:** If advanced verification (`nuXmv`) or graphical LD generation proves too difficult by Day 4.
  - **Action:** We drop them. The verification step will *only* use `matiec` for syntax. The Ladder Logic output will remain the raw, but well-formatted, PLCopen XML. We will re-focus all that extra time on making the core ST generation and the UI absolutely flawless. A perfect "Must-Have" is better than a broken "Should-Have."
- **Plan C: The "Python Override"**
  - **Trigger:** If n8n becomes a bottleneck, is too slow, or proves too inflexible for the complex logic by Day 3.
  - **Action:** The Lead Architect and DevOps specialist pivot. We replace n8n with a simple Python backend using **FastAPI**. It will serve the same purpose (receive requests from Streamlit, call scripts/LLMs) but in a pure-code environment. This is more complex to set up initially but offers maximum performance and control.