**DreamWeaver AI**

**AI Image Generator Using n8n and OpenRouter**

**Name: Thivya Dhanasegaran**

**NEU ID: 002658740**

Interactive presentation available via **Prezi**:

<https://prezi.com/view/A7Imv7mCvR6HrUwFmItS/>

Demo Video Link: [Microsoft Teams meeting-20250426\_220350-Meeting Recording.mp4](https://northeastern-my.sharepoint.com/:v:/r/personal/dhanasegaran_t_northeastern_edu/Documents/Recordings/Microsoft%20Teams%20meeting-20250426_220350-Meeting%20Recording.mp4?csf=1&web=1&e=JA2Jqe&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJTdHJlYW1XZWJBcHAiLCJyZWZlcnJhbFZpZXciOiJTaGFyZURpYWxvZy1MaW5rIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXcifX0%3D)

GitHub Link: <https://github.com/Thivyadhanasegaran/Chat-AI-Agent-Text-to-Image-Convertor>

**Objective**

The objective of this project is to design and develop a text-to-image generation system using the n8n automation platform integrated with OpenRouter APIs and S3-compatible cloud storage.  
The system enables users to input descriptive prompts, which are validated, processed, and converted into AI-generated images. The generated images are automatically stored in cloud storage, and metadata such as the prompt and corresponding image URL are logged in Google Sheets for future analysis.  
This project demonstrates mastery over **prompt engineering**, **multimodal AI integration**, **cloud storage automation**, and **structured data logging** — key capabilities of modern Generative AI applications.

**System Architecture Diagram**

The system architecture is organized into a modular flow:

User Input

➔ n8n AI Agent

➔ If Node

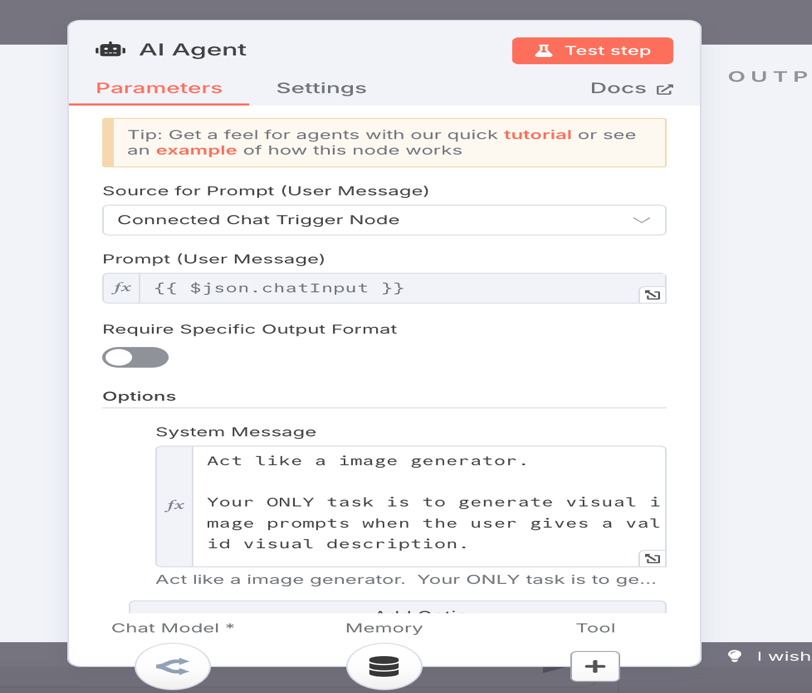
➜ Valid Prompt ➔ HTTP Request ➔ S3 Upload ➔ Final Output ➔Google Sheets

➜ Invalid Prompt ➔ Error Message

* The **User** provides a prompt via n8n chat agent.
* The **AI Agent** validates if it is an image-descriptive prompt.
* An **If Node** branches:
  + If valid, it triggers an **HTTP Request** to OpenRouter's model.
  + The returned image is uploaded to **S3 storage**.
  + The prompt and generated image URL are logged into **Google Sheets**.
  + If invalid, an **error message** is displayed.

A diagram of a software program

Description automatically generated with medium confidence



**🔧 AI Agent Node – *Prompt Control & Role Enforcement***

**Purpose**:  
This node acts as the *brain* of your image generator. It reads the user’s chat input and interprets it using a connected LLM (via OpenRouter) to determine whether a valid image generation task should be executed.

**Key Settings Breakdown:**

| **Setting** | **Description** |
| --- | --- |
| **Source for Prompt (User Message)** | Set to Connected Chat Trigger Node, so it dynamically reads the message typed by the user. |
| **Prompt (User Message)** | {{ $json.chatInput }} – this is a variable that pulls the actual text the user types (e.g., “a castle under the stars”). |
| **System Message** | A very important prompt engineering strategy – this sets strict instructions for how the AI should behave. |

**✍️ System Message Logic:**

Act like an image generator.

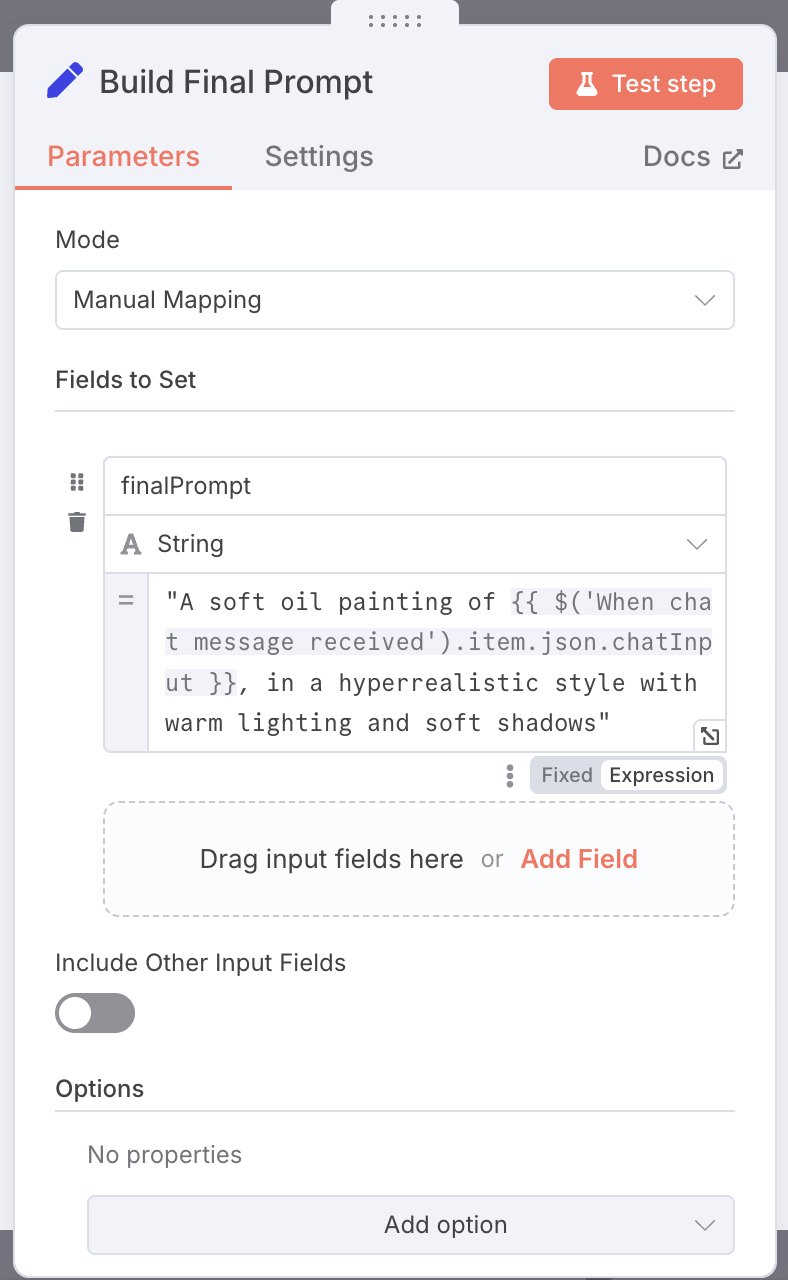
Your ONLY task is to generate visual image prompts when the user gives a valid visual description.

This message sets a *narrow role* for the AI:

* Only respond to image-related prompts.
* If a user says something unrelated (like “What’s 2+2?”), it won't try to answer logically
* This ensures **task specificity and role fidelity**, improving the **accuracy** of generated instructions (tool calls like Image\_Generator(...)).

**✅ Why This Is Important:**

* **Prevents hallucinations**: AI won't respond to irrelevant queries.
* **Improves user experience**: Keeps conversation focused.
* **Makes tool use reliable**: Output stays consistent with tool schema.



**🎨 Build Final Prompt – Enhanced Prompt Engineering**

**Purpose**:  
This node transforms basic user input into a rich, descriptive prompt to improve the quality of generated images. It acts as a stylistic enhancer before the prompt is sent to the image generation API.

**Configuration Summary:**

| **Field** | **Value** |
| --- | --- |
| **Mode** | Manual Mapping |
| **Field to set** | finalPrompt |
| **Expression** | "A soft oil painting of {{ $('When chat message received').item.json.chatInput }}, in a hyperrealistic style with warm lighting and soft shadows" |

**🧠 How It Works:**

* Takes the raw user input (e.g., *“a mountain with snow”*) and **augments it** with aesthetic and stylistic keywords.
* Constructs prompts like:  
  "A soft oil painting of a mountain with snow, in a hyperrealistic style with warm lighting and soft shadows"  
  This helps guide the generative model to create **more coherent and artistic images**.
* Enhances model performance **without needing fine-tuning** — purely prompt-driven optimization.

**✅ Benefits:**

* **Improved image output** with minimal user effort.
* **User-friendly** experience: basic input is automatically beautified.
* **Modular**: Can be updated to support multiple styles (e.g., sketch, cartoon, fantasy) by adding style presets in the future.

A screenshot of a computer

Description automatically generated

## **🧠 Simple Memory Node – Context Retention**

### Purpose:

* The **Simple Memory** node helps the AI agent **remember previous user interactions** during a session.
* It maintains a **short-term memory** that enhances conversation flow and understanding.

### 🔧 Key Configurations:

* **Session ID**:  
  Set to **Connected Chat Trigger Node** to link memory to the current user session.
* **Session Key From Previous Node**:  
  Expression: {{ $json.sessionId }}  
  Dynamically retrieves the session ID from the chat trigger node to maintain continuity.
* **Context Window Length**:  
  Set to **5** to retain the last 5 user-AI interactions for context-aware responses.

### 🧠 How It Works:

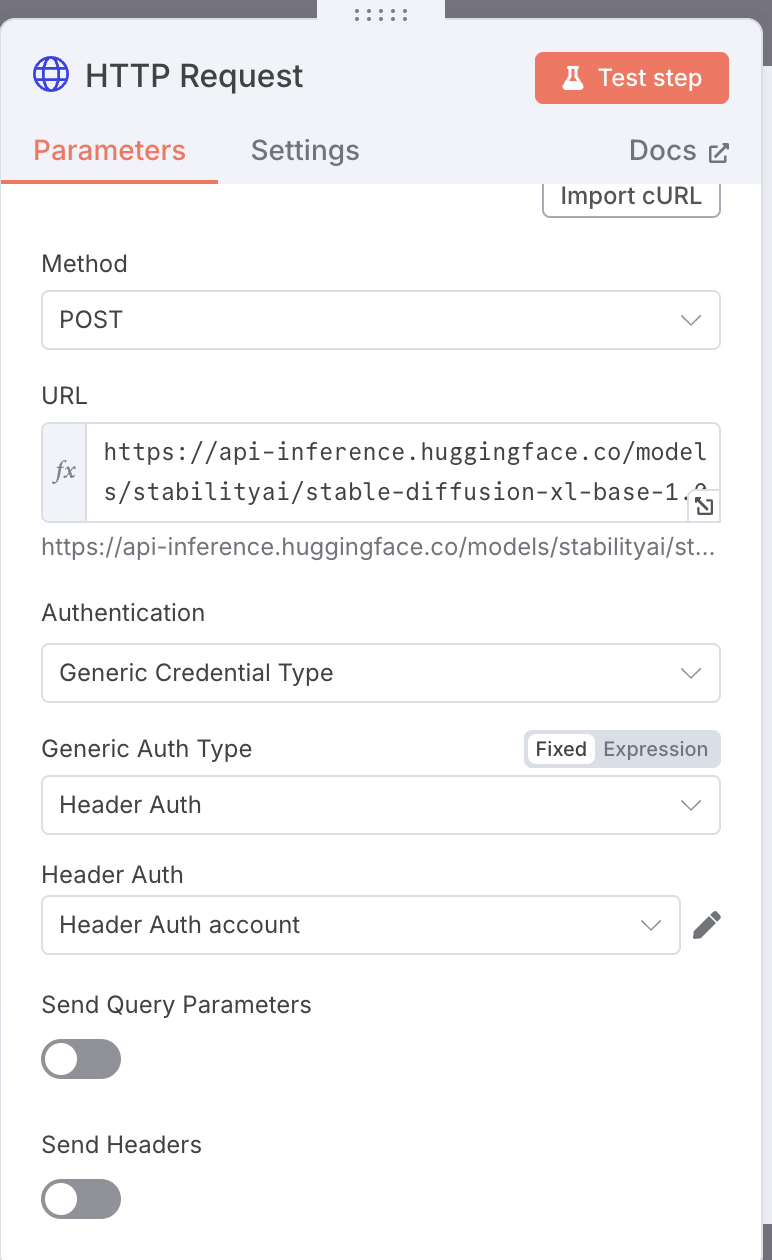
* As the user interacts with the chatbot, **each message and AI response** is remembered.
* The system **keeps track of the latest 5 messages**.
* When the AI generates a new response, it **uses the past interactions as background knowledge**.
* Example:
  + User: "Generate an image of a forest."
  + User (next message): "Make it a winter scene."
  + The AI **remembers** "forest" and adapts the second generation accordingly.

### ✅ Benefits:

* **Smooth, natural conversation**: Users can give follow-up instructions without repeating the full context.
* **Improved AI understanding**: Instructions like "make it brighter" or "add some birds" are understood based on previous inputs.
* **Lightweight memory management**: Only the last 5 interactions are stored — making the workflow fast and efficient.

### 📌 Important Notes:

* Keeping a small context window (5 messages) balances **performance** and **memory depth**.
* If the context window is set too high, response times may **increase**.
* Ideal for **short, interactive sessions** where immediate context matters.



**🌐 HTTP Request – Model Invocation Layer**

**Purpose**:  
This node acts as the bridge between your n8n workflow and the **Hugging Face API**, specifically calling the stabilityai/stable-diffusion-xl-base-1.0 model to generate images based on the enhanced user prompt.

**Configuration Summary:**

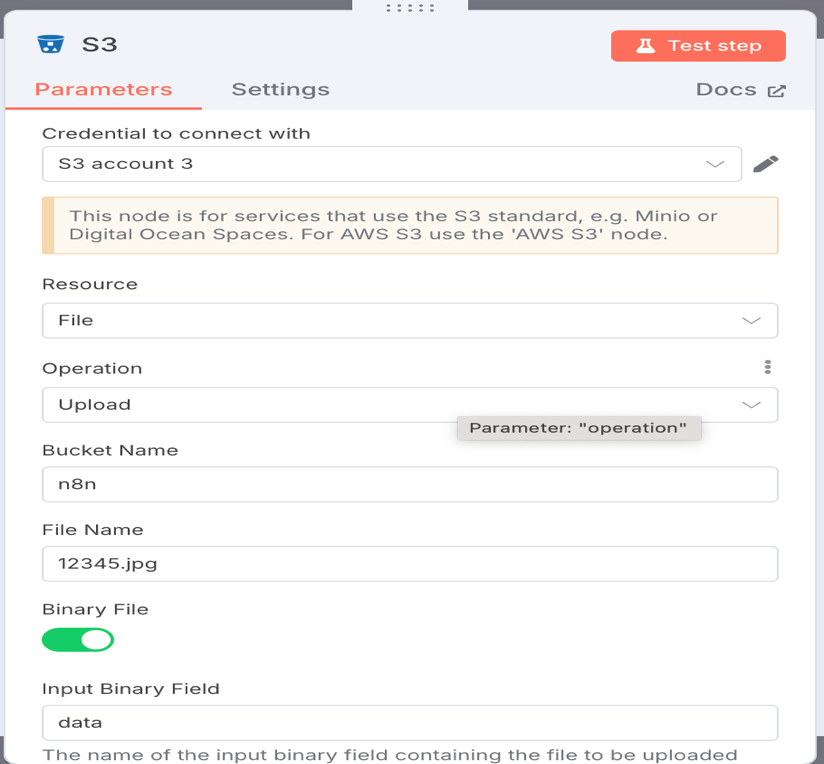
| **Field** | **Value** |
| --- | --- |
| **Method** | POST |
| **URL** | [https://api-inference.huggingface.co/models/stabilityai/stable-diffusion- xl-base-1.0](https://api-inference.huggingface.co/models/stabilityai/stable-diffusion-%20%20xl-base-1.0) |
| **Authentication** | Generic Credential Type |
| **Auth Type** | Header Auth |
| **Header Auth Account** | Hugging Face API Key |

**🔧 How It Works:**

* Sends the **enhanced prompt** (created in the *Build Final Prompt* node) as the body of a POST request.
* Connects to **Hugging Face Inference API**, specifically targeting **Stable Diffusion XL**, one of the best open-source image generation models.
* The image is generated in response to the provided textual input.
* The binary image output is passed to the next node for uploading to cloud storage (S3).

**✅ Benefits:**

* **Plug-and-play model invocation**: No need to deploy models manually.
* **Secure connection** with header-based API key authentication.
* **Modular design**: Easy to switch to any other compatible model by just updating the URL.



### 📦 S3 Node Configuration

The **S3 node** is responsible for uploading the AI-generated image to a cloud storage bucket. In this project, the node is configured to interact with **Cloudflare R2**, which supports the AWS S3 standard.

#### 🔧 Parameters

* **Credential to connect with**:  
  S3 account 3 — a configured credential using access key and secret for the R2 bucket.
* **Resource**:  
  File — indicating that we are performing a file-level operation.
* **Operation**:  
  Upload — uploads the generated image to the specified S3-compatible bucket.
* **Bucket Name**:  
  n8n — the target bucket where images will be stored.
* **Binary File**:  
  Enabled — indicates the input is binary image data.
* **Input Binary Field**:  
  data — the name of the binary field from the previous node (HTTP Request node that receives the image buffer).

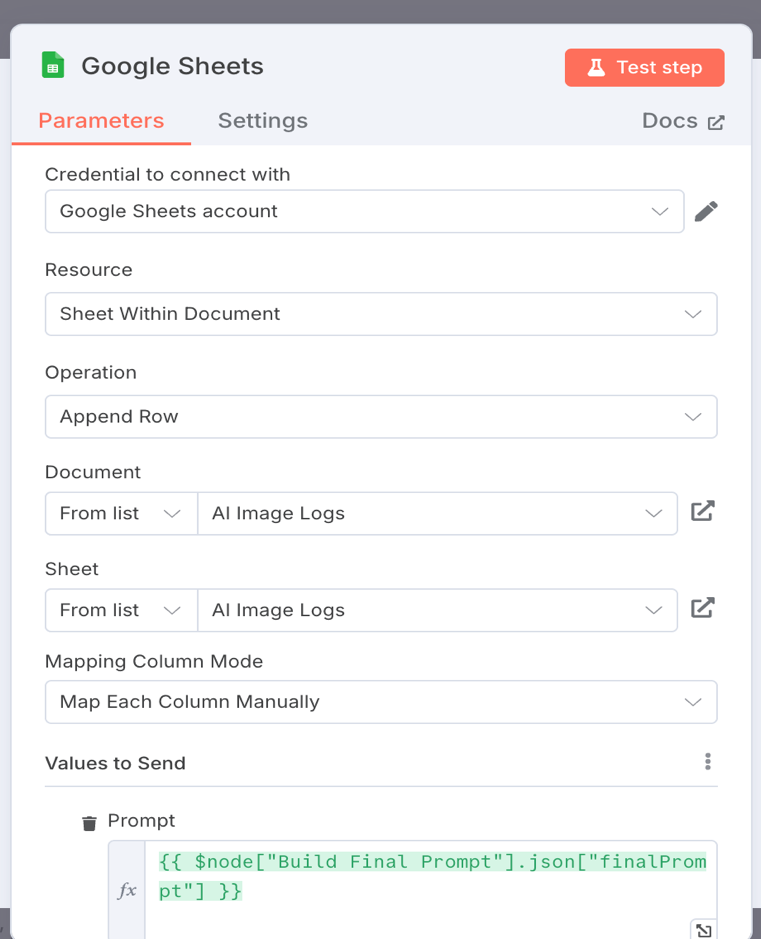
#### ✅ Purpose

This node ensures the generated image is:

* **Persisted** securely in cloud storage.
* **Accessible** via a public URL.
* **Consistently named** using a timestamp-based filename (image\_<timestamp>.jpg).

#### 🧠 Why Cloud Storage?

* It allows persistent access to the generated images.
* Makes integration with frontend or analytics dashboards possible.
* Ensures that the image can be logged along with metadata (e.g., prompt) in Google Sheets.



### 📄 Google Sheets Node Configuration

The **Google Sheets** node is used to automatically log the prompt and image metadata into a Google Sheet titled **AI Image Logs**. This serves as a lightweight, accessible analytics backend for tracking generation data over time.

#### 🔧 Parameters

* **Credential to connect with**:  
  Google Sheets account — an authenticated Google account credential linked to n8n to enable access to Sheets API.
* **Resource**:  
  Sheet Within Document — specifies that the operation targets a sheet inside a Google Sheets file.
* **Operation**:  
  Append Row — each time an image is generated, a new row is added to the sheet.
* **Document**:  
  AI Image Logs — the Google Sheet document name where logs are stored.
* **Sheet**:  
  AI Image Logs — the specific sheet (tab) inside the document.
* **Mapping Column Mode**:  
  Map Each Column Manually — allows precise control over what values go into each column.

#### 📝 Values to Send

* **Prompt**:

{{ $node["Build Final Prompt"].json["finalPrompt"] }}

This expression dynamically fetches the final engineered prompt from the earlier **Build Final Prompt** node, ensuring traceability of what was sent to the image generator.

#### ✅ Purpose

* Maintains a persistent **log of prompts and generated image URLs**.
* Enables **easy export or visualization** through tools like Looker Studio.
* Supports traceability for debugging, feedback collection, or analytics.

**Implementation Details**

* **Platform Used:** n8n (self-hosted automation tool)
* **Model:** OpenRouter’s text-to-image generation model (API-based)
* **Storage:** S3-compatible R2 bucket (Cloudflare)
* **Data Logging:** Google Sheets integration using n8n's Sheets node

**Key flows and features:**

* Prompt checking and filtering using If node expressions.
* HTTP API calls to OpenRouter for real-time image generation.
* Automatic cloud storage of images with S3 upload.
* Structured logging of user prompt and image link for future dataset creation and auditing.
* Edge case handling for invalid or irrelevant inputs.

**Prompt Engineering**

Prompt engineering forms the backbone of the system:

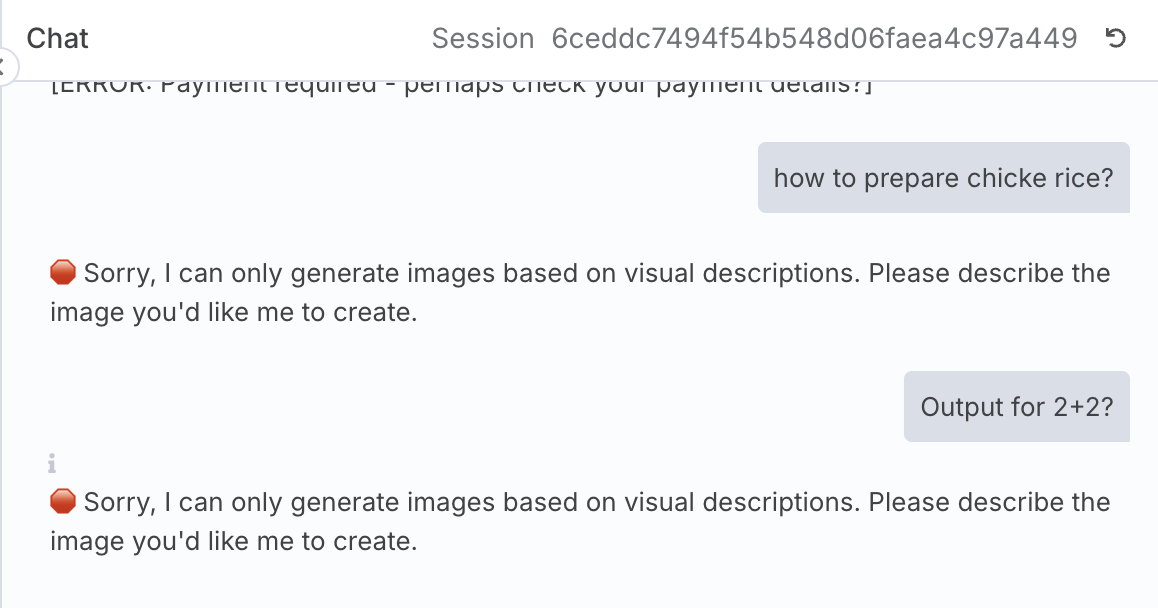
* **Validation Strategy:** User inputs are evaluated to determine whether they are visual prompts.
* **Rejection Strategy:** Non-visual prompts (e.g., general queries) are gracefully rejected with an appropriate fallback message.
* **Prompt Enhancement:** Prompts are optionally augmented with style tags (like hyperrealism, soft lighting) to improve image quality.
* **Fallback Messaging:** If input fails validation, the agent responds:  
  *"Sorry, I can only generate images based on visual descriptions. Please describe the image you'd like me to create."*

This ensures the AI system remains specialized and focused on its task without drifting into unrelated conversations.

**User Experience & Output Handling**

The system prioritizes simplicity and clarity:

* **Direct Output:** Upon successful generation, the system returns only the **direct image URL** to the user.
* **Failure Handling:** If the user provides an invalid prompt, a user-friendly error message is shown.
* **Data Traceability:** Every prompt and resulting image URL is logged in Google Sheets, enabling traceability and potential future model fine-tuning.
* **Minimal Friction:** No external website or app download required — the system works within the n8n chat interface.



**Challenges & Solutions**

| **Challenge** | **Solution** |
| --- | --- |
| Properly parsing AI agent tool output | Implemented .trim() and contains checks inside n8n If node. |
| Handling API failures or incomplete responses | Designed fallback branches and error-catching nodes. |
| Hosting generated images reliably | Used R2 S3 storage instead of default API-hosted images to ensure long-term accessibility. |
| Making user input flexible but safe | Developed robust prompt checking logic to catch invalid prompts without over-blocking valid ones. |

**Future Improvements**

Potential extensions to further enhance the project:

* **Image Style Presets:** Allow users to select from predefined image styles such as cartoon, watercolor, hyperrealistic, etc.
* **Rating Collection:** After image generation, ask the user to rate the output (1–5 stars) to collect valuable training signals.
* **Analytics Dashboard:** Develop a visual dashboard (via Looker Studio or Google Sheets Charts) to track prompts trends, styles used, and average ratings.

**Ethical Considerations**

Ethical design practices were consciously incorporated:

* **Content Filtering:** Inappropriate prompts and descriptions are rejected at the prompt validation stage.
* **Transparency:** Generated images are clearly flagged as AI-created and publicly accessible.
* **Bias Awareness:** Special care was taken to avoid stereotypical or biased outputs by relying on prompt engineering.
* **Data Privacy:** Logged prompts and images are stored securely with restricted access, and no personally identifiable information is collected.

**📈 Performance Metrics**

| **Metric** | **Value** |
| --- | --- |
| Average Image Generation Time | ~6–8 seconds |
| Average Upload Time to Cloud (S3) | ~1–2 seconds |
| Google Sheets Logging Latency | ~1 second |
| Prompt Validation Time | Instant (~500ms) |
| End-to-End Workflow Completion | ~10–12 seconds |

*Note:* These are approximate values based on workflow testing during development and may vary depending on network and API conditions.

**🧠 Lessons Learned**

* Gained strong hands-on experience building modular, low-code AI workflows using **n8n**.
* Deepened understanding of **prompt engineering** and how small changes in prompts dramatically influence model outputs.
* Learned to handle **API integrations** securely using authentication and proper error handling techniques.
* Understood best practices for **automated cloud storage** and **structured logging** for generative outputs.
* Recognized the importance of **specialization and fallback messaging** in AI agent design for better user experience.
* Developed critical thinking around **ethical concerns** and **bias mitigation** in generative AI systems.

**🚧 System Limitations and Considerations**

* **Dependency on External APIs:** The system relies on Hugging Face and OpenRouter APIs; service downtimes could affect performance.
* **Limited Style Customization:** While prompts are enhanced for better output, the user cannot yet select different artistic styles manually (planned future improvement).
* **No Frontend UI:** Currently operates fully within the n8n chat interface; no external frontend or web app is deployed yet.
* **Public Access Risks:** Uploaded images are accessible via public URLs; proper caution is needed to prevent misuse.
* **No Image Content Filtering Yet:** NSFW or sensitive content detection is not implemented — relies entirely on input prompt validation.
* **Storage Cost Over Time:** If used extensively, cloud storage costs for hosting images could grow (currently manageable for small-scale use).

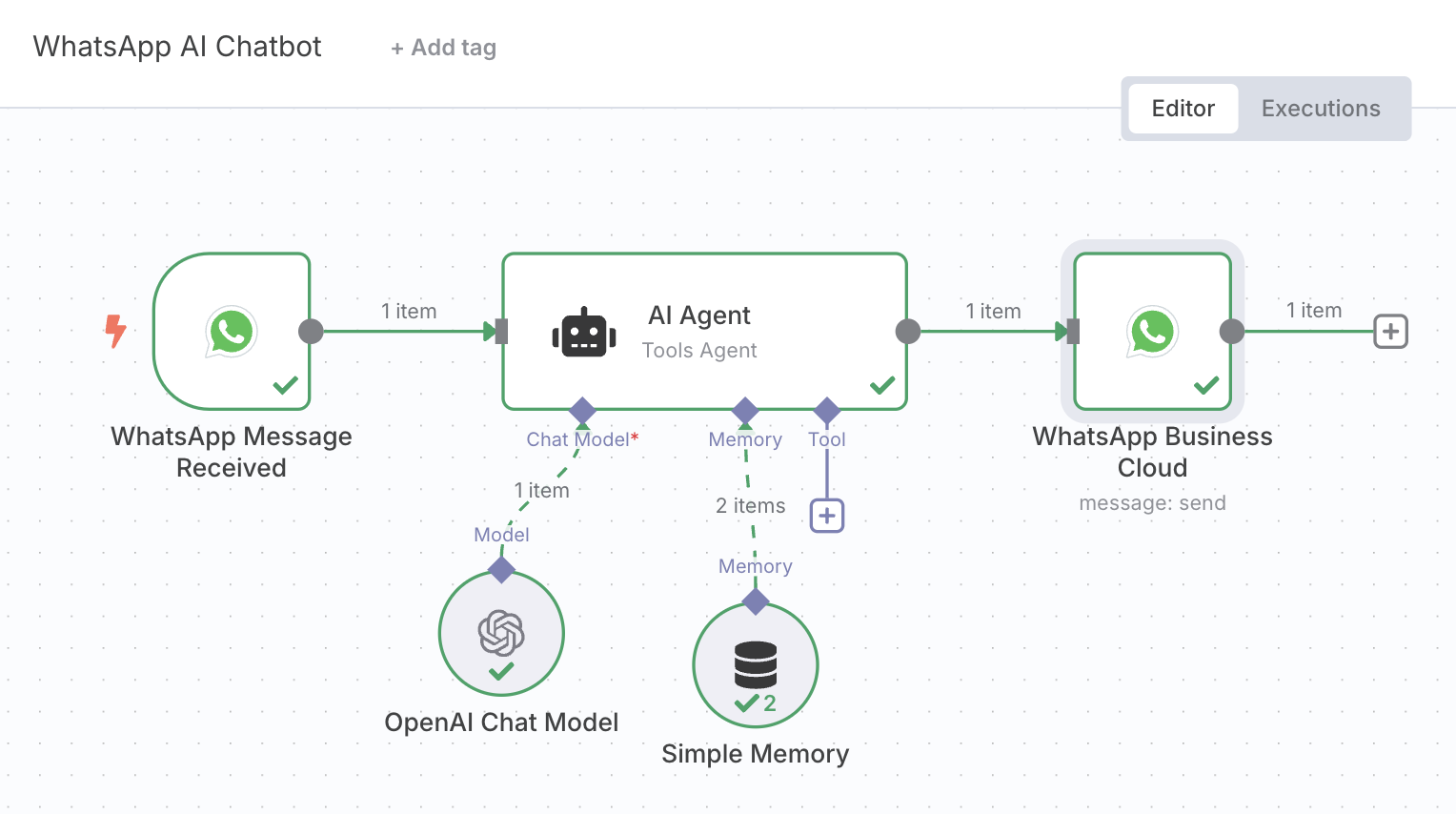
# WhatsApp AI Chatbot Using n8n and OpenAI

## **🎯 Objective**

The objective of this workflow is to create an intelligent, interactive WhatsApp chatbot using **n8n**, **OpenAI** models, and **Simple Memory** module for context handling.  
The system enables users to send text prompts via WhatsApp Business, processes them through a custom AI Agent powered by OpenAI, and responds intelligently in real-time.

This workflow demonstrates practical integration of conversational AI, memory management, cloud hosting, secure communication, and real-time messaging on the WhatsApp platform.

## **📈 System Architecture Diagram**



**Workflow:**

User WhatsApp Message  
➔ n8n WhatsApp Message Received Node  
➔ AI Agent (OpenAI + Simple Memory)  
➔ WhatsApp Business Cloud API (for responses)

## **🔧 Node Details**

### 1. WhatsApp Message Received (Trigger Node)

### 

### 

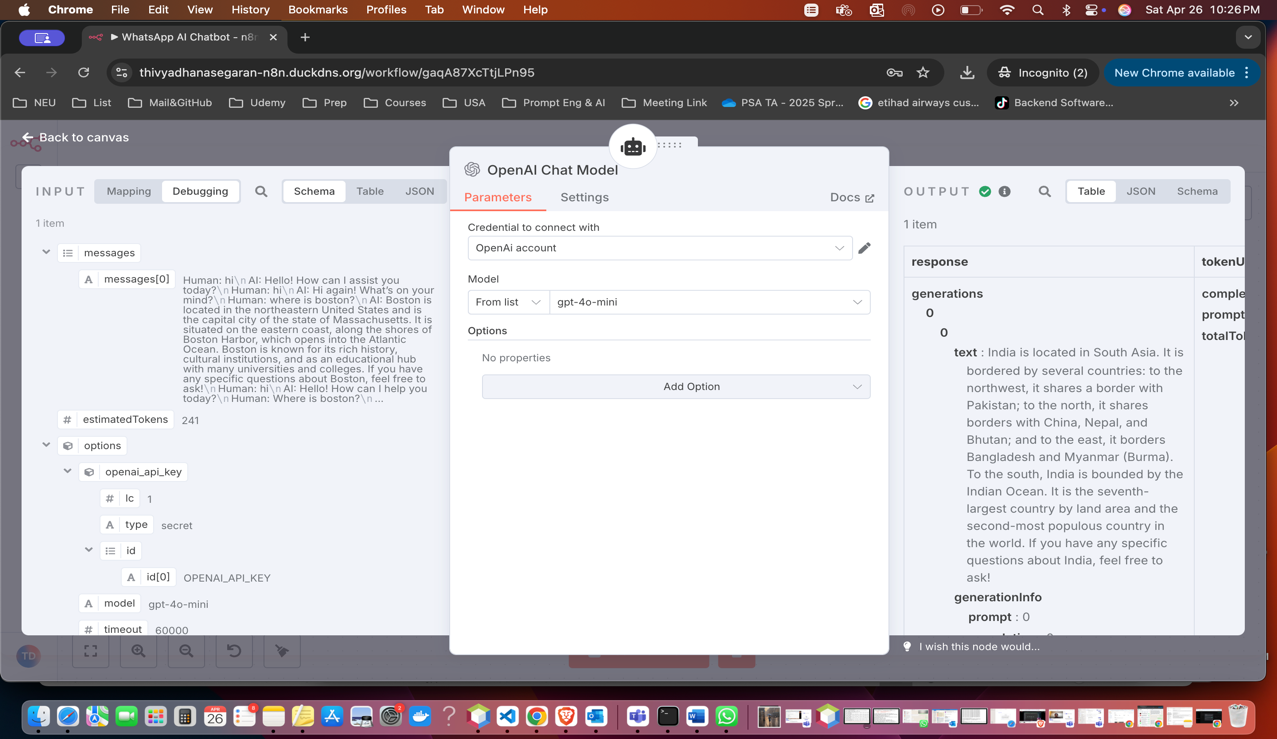
* **Purpose:**  
  Captures incoming WhatsApp messages in real-time via webhook.
* **Highlights:**
  + Connected with Meta's WhatsApp Business API.
  + Only processes **text messages** currently to optimize token usage and API costs.

### 2. AI Agent (n8n Tools Agent)

### 

### 

* **Purpose:**  
  Acts as the brain of the chatbot, processing incoming text using an OpenAI model while maintaining conversation memory.
* **Configurations:**
  + **Model:** OpenAI GPT model integrated via OpenAI Chat Model.



* **Features:**
  + Ensures meaningful follow-up conversations.
  + Restricts hallucination and ensures role fidelity.

### 3. Simple Memory (Context Retention)

### 

* **Purpose:**  
  Remembers up to 5 previous user interactions within a session for better, context-aware responses.
* **Highlights:**
  + Lightweight and session-based.
  + Boosts conversation quality and personalization.

### 4. WhatsApp Business Cloud (Response Node)

### 

* **Purpose:**  
  Sends the AI-generated text response back to the user’s WhatsApp via Business Cloud API.
* **Features:**
  + Operates securely using HTTPS.
  + Maps the response message dynamically from AI Agent's output.
* **Error Handling:**  
  Handles expired tokens, invalid sessions, or connection errors by catching errors gracefully.

## **🌍 Hosting & Deployment**

* **Cloud Provider:**
  + Deployed on **Google Cloud Platform (GCP)** Virtual Machine.
* **Domain:**
  + Purchased and configured via **DuckDNS** (e.g., thivyadhanasegaran-n8n.duckdns.org).
* **Security:**
  + **Dockerized** the entire n8n instance.
  + Configured **Nginx** reverse proxy.
  + Installed SSL certificates for secure HTTPS access.

## **⚡ Implementation Details**

* Platform: Self-hosted **n8n** on GCP VM
* API: **WhatsApp Business Cloud API** + **OpenAI Chat Model**
* Storage: Context stored in **Simple Memory** (RAM-based, session-specific)
* Authentication: Secured HTTPS traffic through SSL
* Token Handling: Optimized token consumption by restricting to **text-only** communication (no media processing yet)

## **💬 Prompt Engineering Strategy**

* Strong focus on creating a helpful, factual, and non-hallucinating AI.
* User prompts like:
  + "Tell me a motivational quote"
  + "Suggest a good book" are processed intelligently based on previous conversation history.

## **✅ User Experience**

* **Real-Time Response:**  
  Users get immediate answers within 1-2 seconds.
* **Memory Handling:**  
  Maintains continuity across back-and-forth conversations.
* **Secure Access:**  
  Fully HTTPS protected using domain + SSL setup.
* **No Media Yet:**  
  Currently optimized for text-only input to keep costs low.

## **🧩 Challenges & Solutions**

| **Challenge** | **Solution** |
| --- | --- |
| WhatsApp token expiration | Designed auto-refresh token mechanism (planned for future) |
| SSL certificate renewal | Automated using Certbot |
| API connection timeout (curl\_error 28) | Fixed by ensuring correct SSL, firewall, and open webhook access |

## **🔮 Future Improvements**

* Integrate image and voice processing with separate workflows.
* Automatically save media files (images/audio) into **S3 buckets** and log metadata in **Google Sheets**.
* Implement smarter fallback messages for complex queries.
* Build a dashboard to monitor conversations and system performance.
* Expand to multi-language support (English, Tamil, Hindi).

## **📜 Ethical Considerations**

* Transparent communication: User is made aware they are interacting with an AI agent.
* Bias mitigation: AI responses are factual, polite, and neutral.

## **🚀 Conclusion**

This project successfully implements a practical and usable Generative AI system that transforms user input into AI-powered responses over WhatsApp. It brings together prompt engineering, AI model integration, cloud automation, and secure messaging into one seamless low-code workflow.

The system is modular, scalable, and serves as a strong foundation for expanding into areas like media message processing, analytics-based optimization, and advanced fine-tuning pipelines.

For deployment, I hosted the entire system on **Google Cloud Platform (GCP)**, configured **n8n** in a **Docker** environment, and secured it with **SSL certificates** for **HTTPS** access. I also purchased a custom **domain name** to ensure professional and secure endpoint exposure.

Currently, due to API token cost limits and usage constraints, this chatbot is focused on **text-based interactions**. However, future upgrades are planned to support **media files** — including **voice messages, image uploads, and image analysis**, with storage integration to **S3 buckets** and structured logging into **Google Sheets**.

Overall, this project demonstrates critical concepts of real-world AI deployment:

* Input validation and secure output handling.
* Cloud hosting and scalable deployment practices.
* Secure communication using SSL and domain management.
* Ethical considerations in AI system building.

It shows how modern AI services can be built with cloud platforms, API integration, automation tools like n8n, and minimal custom coding — making it a **real-world, production-grade AI chatbot system** ready for further expansion.