Summary: This code handles logic to work with webhook

```
try {
        const response = await fetch("https://vansh-tyagi.app.n8n.cloud/webhook-test/ai-image", {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({ text: userMessage }),
        });

        let rawText = await response.text();
        console.log("📥 Raw response:", rawText);
```

The CSS part is mention over here:

```
<style>
    body {
      font-family: Arial, sans-serif;
      padding: 1rem;
      background-color: #f0f0f0;
    }

    h1 {
      margin-bottom: 1rem;
    }

    #chat {
      max-width: 600px;
      margin: auto;
      background: white;
      padding: 1rem;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0,0,0,0.1);
    }

    #chatBox {
      max-height: 400px;
      overflow-y: auto;
      margin-bottom: 1rem;
    }

    .message {
      margin-bottom: 1rem;
      line-height: 1.5;
    }

    .message.user {
      color: green;
      text-align: right;
    }

    .message.bot {
      color: #333;
      text-align: left;
    }

    input {
      padding: 0.5rem;
      width: 70%;
      font-size: 16px;
    }

    button {
      padding: 0.5rem;
```

Selector/class	What It Styles	Main Properties & Effects
body	Entire page	Font, padding, background
h1	Main heading	Space below
#chat	Chat container	Width, centering, background, padding, shadow
#chatBox	Message area	Height, scroll, space below
.message	All messages	Space below, line height
.message.user	User messages	Green color, right-aligned
.message.bot	Bot messages	Dark gray, left-aligned
input	Input field	Padding, width, font size

```
    font-size: 16px;
    background-color: #2b8a3e;
    color: white;
    border: none;
    cursor: pointer;
  }

  button:hover {
    background-color: #256e33;
  }
</style>
```

button	Button	Padding, font size, color, no border
button:hover	Button (hover)	Darker green color

This is the frontend part of html, basically objects of the page:

```
<body>
  <div id="chat">
    <h1>Ask Perplexity 🤖</h1>
    <div id="chatBox"></div>
    <input id="userInput" type="text"
placeholder="Type your message..." />
    <button
onclick="sendMessage()">Send</button>
  </div>
```

Element	Purpose
`<body>`	Main container for the webpage content
`<div id="chat">`	Contains all chat elements
`<h1>Ask Perplexity 🤖</h1>`	Displays the chat title
`<div id="chatBox"></div>`	Displays chat messages
`<input id="userInput" ... />`	User types messages here
`<button onclick="sendMessage()">`	Sends the user's message when clicked

Now the Script code that is handling all the logic:

Code blocks : from 1st till last	Reasons
`const inputField = document.getElementById("userInput");` `const chatBox = document.getElementById("chatBox");`	Explanation: These lines grab two HTML elements by their IDs. • inputField is likely a text input where users type messages. • chatBox is the container where chat messages are displayed.
`function appendMessage(sender, text, className) {`	Explanation: • This function adds a new message to the chat box. • It takes three arguments: sender: Who sent the message (e.g., "You" or "Bot") text: The message content className: A CSS class for styling (e.g., "user", "bot")
`const msgDiv = document.createElement("div");` `msgDiv.className = `message ${className}`;`	Explanation: • msgDiv is a new `<div>` element created for the message. • className adds classes for styling (e.g., `"message bot"`).
`// Replace \n with   for bot responses only` `if (className === "bot") {` `text = text.replace(/\n/g, " ");` `msgDiv.innerHTML = `${sender}: ${text}`;` `} else {` `msgDiv.innerHTML = `${sender}: ${escapeHTML(text)}`;` `}`	Explanation: • If it's a bot message (`className === "bot"`): • Newlines (`\n`) in the message are replaced with ` ` tags to display line breaks. • The message is formatted with the sender in bold, followed by a line break, then the message. • If it's not a bot message: • The message uses `escapeHTML(text)` (a function not shown here, but typically used to prevent XSS by escaping HTML special characters). • The message is formatted with the sender in bold, then the message text.
`chatBox.appendChild(msgDiv);` `chatBox.scrollTop = chatBox.scrollHeight;`	Explanation: • appendChild(msgDiv): Adds the new message to the chat box. • scrollTop = scrollHeight: Scrolls the chat box to the bottom, so the latest message is visible.

`function escapeHTML(str) {`
`return str.replace(/[&<>"']/g, (m) => ({`
`"&": "&",`
`"<": "<",`
`">": ">",`
`'"': """,`
`"'": "'"`
`}[m]));`
`}`
	1. `/[&<>"']/g`
	  • `[&<>"']` matches any of the characters: `&`, `<`, `>`, `"`, or `'`.
	  • `g` means "global"—it will find all matches in the string, not just the first one.
	2. `(m)` is the matched character (e.g., `"<"`).
	• `({ ... }[m])` looks up the matched character in an object and returns its HTML entity equivalent.
	  • `"&"` becomes `"&"`
	  • `"<"` becomes `"<"`
	  • `">"` becomes `">"`
	  • `'"'` (double quote) becomes `"""`
	  • `"'"` (single quote) becomes `"'"`
	• Prevents HTML injection:
	If someone tries to insert HTML or JavaScript, it will be displayed as text, not executed.
	• Keeps your app secure:

This is a basic but important security measure.

```javascript
  async function sendMessage() {
    const userMessage = inputField.value.trim();
    if (!userMessage) return;

    appendMessage("You", userMessage, "user");
    inputField.value = "";

    try {
      const response = await
fetch("https://vansh-tyagi.app.n8n.cloud/webhook-test/ai-image", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ text: userMessage }),
      });

      let rawText = await response.text();
      console.log("📥 Raw response:", rawText);

      // 🧹 Remove <think>...</think>
      rawText = rawText.replace(/<think>[\s\S]*?<\/think>/gi,
"").trim();

      // 🧹 Remove outer { ... } if present
      if (rawText.startsWith("{") && rawText.endsWith("}")) {
        rawText = rawText.slice(1, -1);
      }

      //remove first occurance of "mesaages:"
      rawText = rawText.replace(/["']?message["']?\s*:\s*/, "");


      // 🧹 Remove all asterisks used for markdown
      rawText = rawText.replace(/\*/g, "");

      appendMessage("Bot", rawText, "bot");

    } catch (error) {
      console.error("❌ Fetch Error:", error);
      appendMessage("Bot", `❌ Network Error: ${error.message}`,
"bot");
    }
  }
```

1. `async function sendMessage() {`

- `async` means this function can use `await` to handle promises (like network requests).
- Purpose: Sends a user's message and handles the bot's response.

2. Get User Input

```javascript
const userMessage = inputField.value.trim();
if (!userMessage) return;
```

- `inputField.value.trim()`:
  - Gets the text the user typed and removes extra spaces.
- `if (!userMessage) return;`:
  - If the input is empty, the function stops here.

3. Display User Message

```javascript
appendMessage("You", userMessage, "user");
inputField.value = "";
```

- `appendMessage(...)`:
  - Adds the user's message to the chat box.
- `inputField.value = ""`:
  - Clears the input field for the next message.

---

4. Send Message to Server

```javascript
try {
  const response = await
fetch("https://vansh-tyagi.app.n8n.cloud/webhook-test/ai-image", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ text: userMessage }),
  });
```

- `fetch(...)`:
  - Sends a POST request to the server with the user's message.

- `Content-Type: application/json`:
  - Tells the server the data is JSON.
- `body: JSON.stringify({ text: userMessage })`:
  - Sends the message as `{ "text": "your message" }`.
- `JSON.stringify()` is a JavaScript function that **converts a JavaScript object or value into a JSON-formatted string**.
- **Headers** in the context of HTTP (the protocol used for web communication) are key-value pairs of information sent with every request from a client (like a browser) to a server, and with every response from the server back to the client

These headers provide important metadata about the request or response, such as:

- Content type (e.g., `Content-Type: application/json` tells the server or client what format the data is in)
- Authentication (e.g., `Authorization: Bearer ...` for login tokens)
- Caching instructions (e.g., `Cache-Control: no-cache`)
- Language preferences (e.g., `Accept-Language: en-US`)
- Request origin and user agent (e.g., `User-Agent` and `Referer`)
- Headers are used to help both sides understand how to process the request or response, manage security, control caching, and much more

5. Handle Server Response

```javascript
let rawText = await response.text();

console.log("📥 Raw response:", rawText);
```

- `response.text()`:
  - Gets the server's response as text.
- `console.log(...)`:
  - Logs the raw response for debugging.

---

6. Clean Up the Response

```javascript
// 🧹 Remove <think>...</think>

rawText = rawText.replace(/<think>[\s\S]*?<\/think>/gi, "").trim();
```

```javascript
// 🧹 Remove outer { ... } if present
if (rawText.startsWith("{") && rawText.endsWith("}")) {
  rawText = rawText.slice(1, -1);
}

// remove first occurrence of "message:"
rawText = rawText.replace(/["']?message["']?\s*:\s*/, "");

// 🧹 Remove all asterisks used for markdown
rawText = rawText.replace(/\*/g, "");
```

- Removes `<think>...</think>`:
  - Deletes any text between `<think>` and `</think>`.
- Removes outer curly braces `{ ... }`:
  - If the response is wrapped in curly braces, removes them.
- Removes `"message:"`:
  - Removes the first occurrence of `message:` (with optional quotes and spaces).
- Removes asterisks `*`:
  - Removes all asterisks (often used for markdown formatting).

---

7. Display Bot Response

```javascript
appendMessage("Bot", rawText, "bot");
```

- `appendMessage(...)`:
  - Adds the cleaned-up bot response to the chat box.

8. Error Handling

```javascript
} catch (error) {
  console.error("❌ Fetch Error:", error);
  appendMessage("Bot", `❌ Network Error: ${error.message}`, "bot");
}
```

- `catch (error)`:
  - Catches any errors during the fetch or processing.
- `console.error(...)`:
  - Logs the error.
- `appendMessage(...)`:
  - Shows an error message in the chat box.

```javascript
// Optional: Trigger send on Enter key
inputField.addEventListener("keypress", (e) => {
  if (e.key === "Enter") sendMessage();
});
```

Code Part	What It Does
`inputField.addEventListener("keypress", ...)`	Listens for key presses in the input field
`(e) => { ... }`	Runs code when a key is pressed
`if (e.key === "Enter")`	Checks if the Enter key was pressed
`sendMessage();`	Calls the send message function

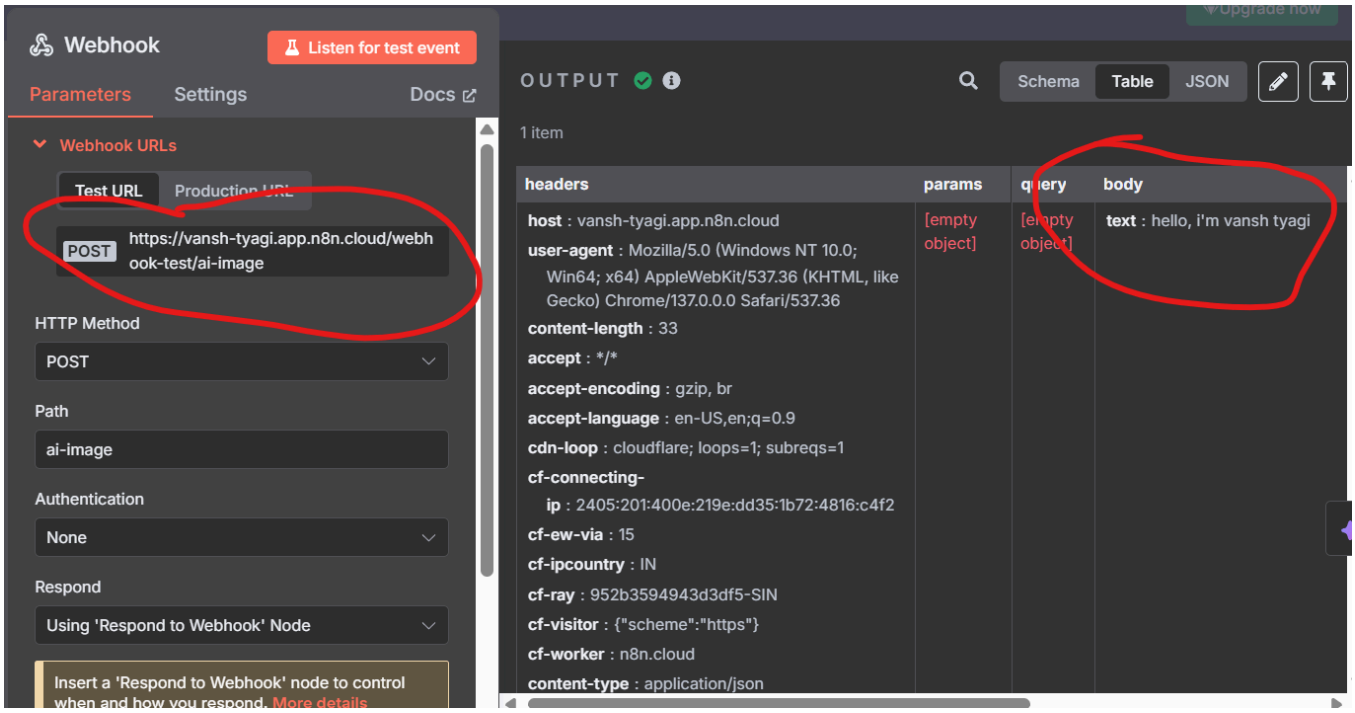
Ask Perplexity 🤖

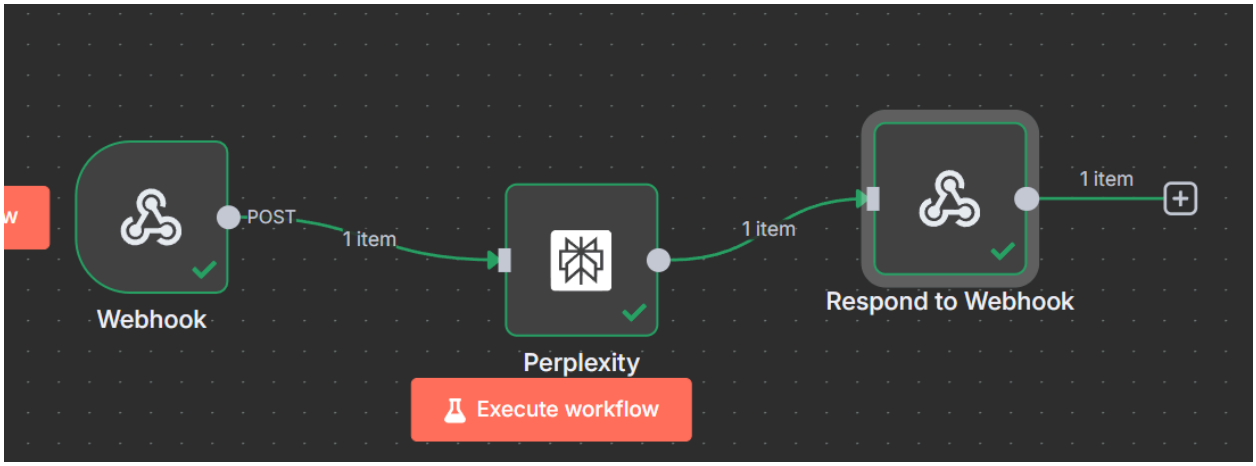
You: hello, i'm vansh tyagi

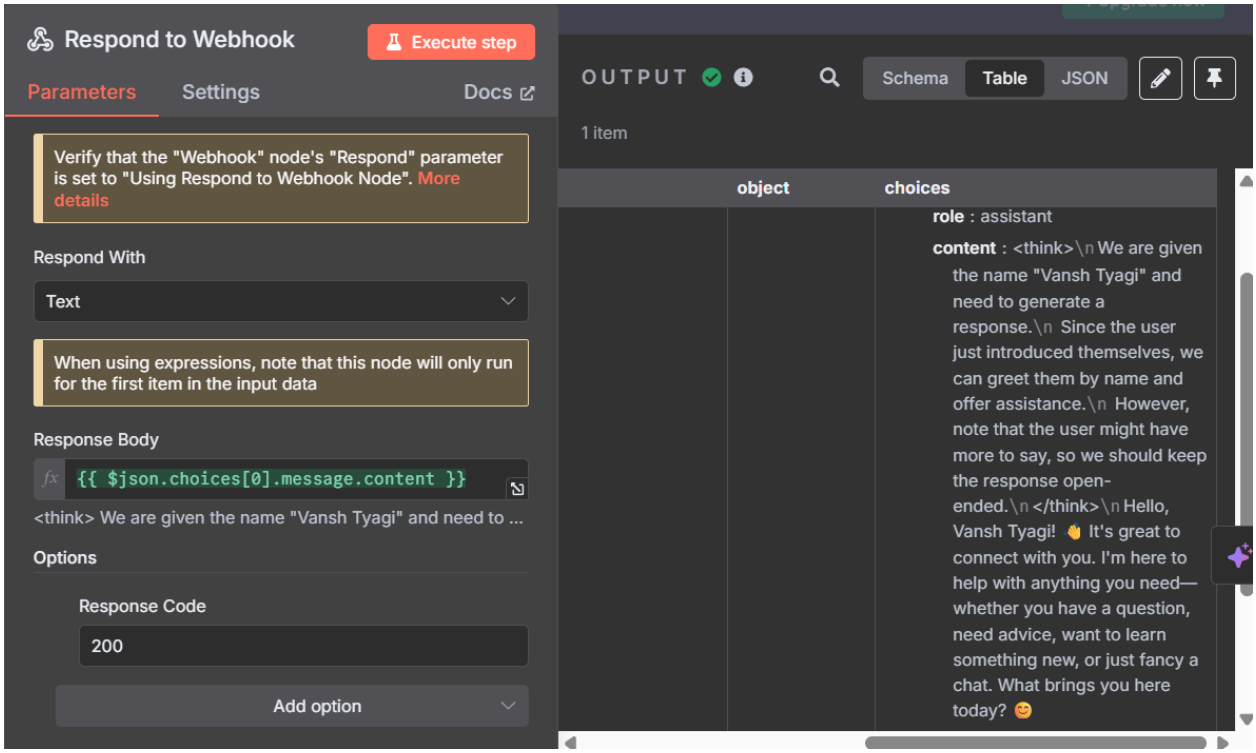
Bot:
Hello, Vansh Tyagi! 👋 It's nice to meet you. Is there something I can help you with today? Whether you have a question, need advice, or just want to chat—I'm here for it! 😊

Let me know what brings you here!

Type your message... Send




Webhook
POST
1 item
Perplexity
1 item
Respond to Webhook
1 item
Execute workflow


Respond to Webhook
Execute step
Parameters
Settings
Docs
Verify that the "Webhook" node's "Respond" parameter is set to "Using Respond to Webhook Node". More details
Respond With
Text
When using expressions, note that this node will only run for the first item in the input data
Response Body
{{ $json.choices[0].message.content }}
<think> We are given the name "Vansh Tyagi" and need to ...
Options
Response Code
200
Add option
OUTPUT
Schema
Table
JSON
1 item
object
choices
role : assistant
content : <think>\n We are given the name "Vansh Tyagi" and need to generate a response.\n Since the user just introduced themselves, we can greet them by name and offer assistance.\n However, note that the user might have more to say, so we should keep the response open-ended.\n </think>\n Hello, Vansh Tyagi! 👋 It's great to connect with you. I'm here to help with anything you need—whether you have a question, need advice, want to learn something new, or just fancy a chat. What brings you here today? 😊