### **Chat History Workflow**

The core idea is that your client application will maintain a list of all messages exchanged. For every new question the user asks, the client sends the new question *plus the entire history* to the API.

**Initialize History:** When a user starts a chat, your application creates an empty array in its state to store the conversation.  
JavaScript  
let chatHistory = [];

1. **User Asks a Question:** The user types their first question, for example, "Who was the farmer?".

Your app adds this question to the chatHistory array.  
JavaScript  
// Add human message to history

chatHistory.push({ role: 'human', content: 'Who was the farmer?' });

1. **Call the API:** Your application now calls your POST /agent/product-chatbot endpoint. The request body includes the product data, the new question, and the current chat history.

**Request Body:**JSON  
{

"product\_data": { ... },

"question": "Who was the farmer?",

"chat\_history": [

{ "role": "human", "content": "Who was the farmer?" }

]

}

1. **Receive and Store Response:** The API processes the request and returns the AI's answer, for example, {"answer": "This product was grown by Rajesh Kumar."}.

Your app adds this AI response to the chatHistory array.  
JavaScript  
// Add AI response to history

chatHistory.push({ role: 'ai', content: 'This product was grown by Rajesh Kumar.' });

* + The chatHistory array now contains two messages.

1. **User Asks a Follow-up Question:** The user then asks, "Where is his farm located?".
   * Your app adds this new question to the chatHistory array. It now has three messages.
   * It calls the API again, but this time, it sends the **complete, updated chat history**.

**Request Body:**JSON  
{

"product\_data": { ... },

"question": "Where is his farm located?",

"chat\_history": [

{ "role": "human", "content": "Who was the farmer?" },

{ "role": "ai", "content": "This product was grown by Rajesh Kumar." },

{ "role": "human", "content": "Where is his farm located?" }

]

}

The backend uses this rich history to understand the context ("his farm" refers to Rajesh Kumar's farm) and provide an accurate follow-up answer. This cycle repeats for the entire conversation.

### **Example Frontend Code (JavaScript)**

Here is a simple JavaScript example demonstrating how to implement this logic.

JavaScript

// This would be part of your frontend application's state

let chatHistory = [];

const productData = { /\* ... your full product JSON ... \*/ };

const apiUrl = 'https://your-api-url.onrender.com/agent/product-chatbot';

// Function to handle sending a message and updating history

async function sendMessage(question) {

// 1. Add the user's message to our local history

chatHistory.push({ role: 'human', content: question });

// (Here, you would update your UI to show the user's message)

try {

// 2. Prepare the request body with the full history

const requestBody = {

product\_data: productData,

question: question,

chat\_history: chatHistory

};

// 3. Call the API

const response = await fetch(apiUrl, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify(requestBody),

});

if (!response.ok) {

throw new Error(`API Error: ${response.statusText}`);

}

const data = await response.json();

const aiAnswer = data.answer;

// 4. Add the AI's response to our local history

chatHistory.push({ role: 'ai', content: aiAnswer });

// (Here, you would update your UI to show the AI's answer)

} catch (error) {

console.error("Failed to get response from chatbot:", error);

// (Optionally show an error message in the UI)

}

}

// --- How to use it ---

// sendMessage("What is the curcumin content?");

// After the first response, you can ask a follow-up:

// sendMessage("Is that considered a good amount?");

No changes are needed to your Python backend. Its current design correctly supports this client-side history management.