**Comprehensive Explanation Summary**

**1️⃣ Project Overview**

“We implemented a system where a user can type a query in an Angular frontend and get intelligent answers based on our Excel dataset. The system combines **Azure Cognitive Search** for searching the dataset and **Azure GPT** for generating natural language responses.”

* **Frontend:** Angular application where user enters prompts.
* **Backend:** Node.js Express server handles search and GPT integration.
* **Dataset:** Excel file stored in Azure Blob Storage.
* **Search Engine:** Azure Cognitive Search indexes the Excel dataset.
* **AI Model:** Azure OpenAI GPT generates answers using search results as context.

**2️⃣ Data Preparation**

* **Step 1:** Created an Excel dataset with product information (ProductName, Price, Category, Brand).
* **Step 2:** Uploaded the Excel file to **Azure Blob Storage**.
* **Step 3:** Created an **index** in Azure Cognitive Search to make the dataset searchable.
* **Step 4:** Configured **data source → index → indexer** so Cognitive Search can fetch and index Excel data.

**Key point:** Each row in Excel becomes a **searchable document**, which GPT can later use as context.

**3️⃣ Why Node.js for Backend**

“We chose Node.js for several reasons:”

1. **JavaScript/TypeScript compatibility** with Angular frontend → seamless integration.
2. **Asynchronous handling** → can call Azure Search and GPT APIs simultaneously without blocking.
3. **Quick to set up REST APIs** → easy to create /api/query route for frontend communication.
4. **Rich SDK support** → Azure provides Node.js SDKs for Cognitive Search and GPT integration.

**4️⃣ Backend Implementation**

**Step 1: API Route**

* Created /api/query POST route in Node.js.
* Receives **user prompt** from frontend.

app.post('/api/query', async (req, res) => {

const userPrompt = req.body.prompt;

});

**Step 2: Query Azure Cognitive Search**

* Used @azure/search-documents SDK.
* Backend connects using **endpoint URL**, **index name**, and **API key**.
* Fetches top N matching documents based on user prompt.

const searchResults = await searchClient.search(userPrompt, { top: 10 });

const documents = [];

for await (const result of searchResults.results) {

documents.push(result.document);

}

**Key point:** Search results are **structured and include fields from Excel**.

**Step 3: Format Search Results for GPT**

* Backend converts search results into readable context for GPT:

const context = documents.map(doc =>

`Product: ${doc.ProductName} | Price: ${doc.Price} | Category: ${doc.Category}`

).join("\n");

**Step 4: Call Azure GPT**

* Backend sends **prompt + context** to GPT using Azure OpenAI endpoint:

const gptPrompt = `Answer ONLY using the following context:\n${context}\nQuestion: ${userPrompt}`;

const response = await fetch("<Azure-GPT-endpoint>", {

method: "POST",

headers: { "api-key": "<key>", "Content-Type": "application/json" },

body: JSON.stringify({ messages: [{ role: "system", content: "Answer using only the context" }, { role: "user", content: gptPrompt }] })

});

const gptData = await response.json();

**Key point:** GPT is instructed to **use only the dataset context**.

**Step 5: Send Response to Frontend**

* Backend returns GPT output as JSON:

res.json({ response: gptData.choices[0].message.content });

* Frontend displays it in **Submitted Prompts section**.

**5️⃣ Frontend Flow (Angular)**

1. User types a prompt → clicks **Submit**.
2. Angular sends prompt to backend via fetch or HttpClient.
3. Backend queries Cognitive Search + GPT.
4. Response is displayed below input area.

**UI shows:** Prompt → GPT answer → allows clearing input for next query.

**6️⃣ Why GPT Sometimes Gives Generic Answers**

1. **Limited or no search results**
   * If Cognitive Search returns empty or very few documents, GPT fills the answer using its own knowledge.
2. **Context formatting issues**
   * If the backend does not format search results clearly, GPT may not use them properly.
3. **Default GPT behavior**
   * GPT always tries to give a useful answer even if context is limited.

**Solution:**

* Ensure backend sends **structured context from search results**.
* Use **system message**: “Answer only using the provided context.”
* Increase top parameter in search to include more dataset rows.

**7️⃣ End-to-End Flow Summary**

[Angular Frontend]

|

v

[Node.js Backend] --> Queries [Azure Cognitive Search] --> Returns results

| |

v |

Sends context + prompt ------------------------|

|

v

[Azure GPT] --> Generates response

|

v

[Backend returns JSON]

|

v

[Frontend displays answer]

**✅ Key Points to Highlight**

* Node.js backend acts as **middleman** for secure and structured integration.
* GPT does **not directly access Excel**; it uses **search results as context**.
* Cognitive Search ensures **answers are based on actual dataset**.
* Proper prompt formatting ensures GPT **does not hallucinate**.
* System now ready for **frontend demonstration** with real queries.