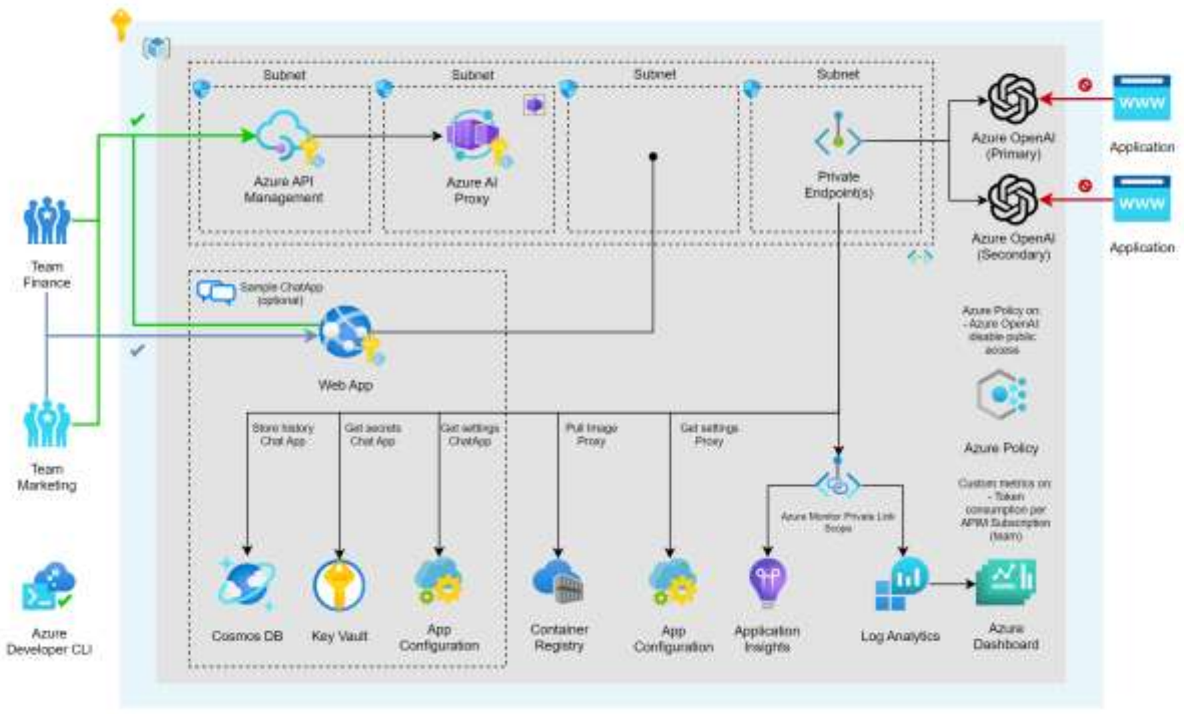
Integrating Azure OpenAI, with **ASP.NET Core + Blazor (Server or WASM)**

---

1 Setting up an Azure OpenAI Service



What is Azure OpenAI?

Azure OpenAI Service provides enterprise access to OpenAI models (GPT-4.x, GPT-4o, embeddings, etc.) with:

- Azure AD & API key security
- Regional compliance
- Private networking
- SLA & enterprise governance

Step-by-Step Setup

1. Create the Azure OpenAI resource

- Azure Portal → **Create Resource**

- Search **Azure OpenAI**
- Select region (important for latency + compliance)
- Pricing tier: *Standard S0*

2. Deploy a model

Azure OpenAI **requires model deployment names**.

Example:

Model	Deployment Name
GPT-4o-mini	gpt-chat
text-embedding-3-large	embeddings

⚠️ **You never call the model name directly—only the deployment name**

Required Configuration

```
// appsettings.json
{
  "AzureOpenAI": {
    "Endpoint": "https://your-openai-resource.openai.azure.com/",
    "ApiKey": "YOUR_API_KEY",
    "ChatDeployment": "gpt-chat",
    "EmbeddingDeployment": "embeddings"
  }
}
```

2️⃣ Implementing Smart Pasting (AI-assisted paste)

What is Smart Pasting?

When a user pastes **raw or messy content**, AI:

- Cleans formatting
- Summarizes
- Converts to structured text (Markdown / JSON / bullets)
- Removes sensitive data

Architecture

Clipboard → Blazor UI

→ API → Azure OpenAI

→ Cleaned Output → UI

Blazor Demo – Smart Paste

Razor Component

```
<textarea @onpaste="HandlePaste" class="form-control" rows="6"></textarea>


<p><b>AI Enhanced:</b></p>

<textarea class="form-control" rows="6">@enhancedText</textarea>
```

Code-Behind

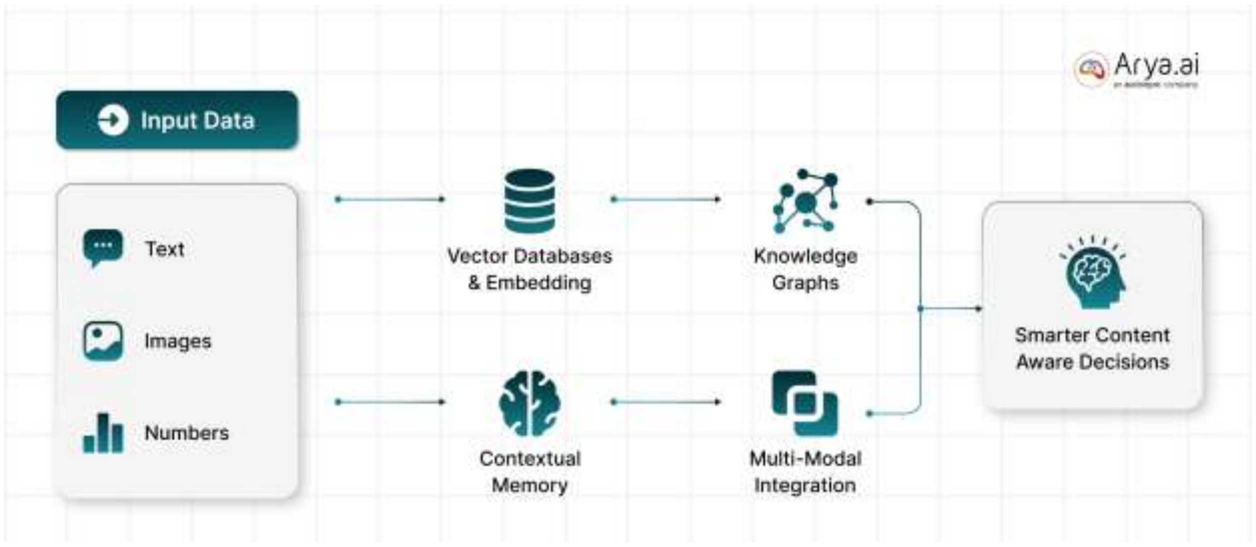
```
private string enhancedText;


private async Task HandlePaste(ClipboardEventArgs e)
{
    var pastedText = await JS.InvokeAsync<string>(
        "navigator.clipboard.readText");


    enhancedText = await _aiService.SmartPasteAsync(pastedText);
}
```

AI Prompt Logic

```
public async Task<string> SmartPasteAsync(string input)
{
    var prompt = $"""
    Clean and structure the following text.
    Remove noise, fix grammar, and return concise bullet points.

    Text:
    {input}
    """;

    return await SendToOpenAI(prompt);
}
```

---

3 Implementing a Smart Text Area (Context-Aware AI Writing)



What Makes a Text Area “Smart”?

- Suggests next sentence
- Improves tone
- Auto-completes based on **context**
- Grammar + clarity optimization

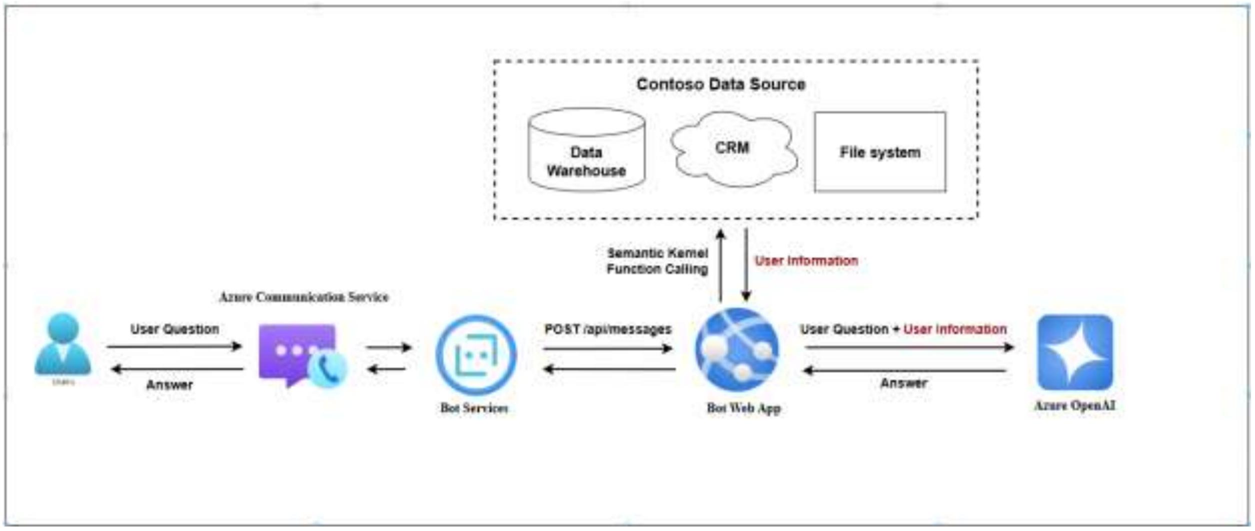
Smart Text Area Example

```
<textarea @bind="userText" class="form-control"></textarea>

<button @onclick="ImproveText">Improve</button>


<p><b>Suggestion:</b> @aiSuggestion</p>

private async Task ImproveText()

{

   aiSuggestion = await _aiService.EnhanceTextAsync(userText);

}

public async Task<string> EnhanceTextAsync(string text)

{

   var prompt = $"""

   Improve clarity and professionalism.

   Do not change meaning.


   Text:

   {text}

   """;


   return await SendToOpenAI(prompt);
}
```

📌 Use cases

- Email drafting
- Code comments
- Requirement descriptions
- User stories (Jira 👍)

4️⃣ Adding a ChatBot (Conversational AI)



Core Concepts

- **System Prompt** → defines bot personality
- **Conversation Memory**
- **User Messages**
- **Assistant Responses**

ChatBot UI (Blazor)

```
<input @bind="userMessage" />
```

```
<button @onclick="Send">Send</button>

@foreach (var msg in chatHistory)
{
   <p><b>@msg.Role:</b> @msg.Content</p>
}
private async Task Send()
{
   var reply = await _aiService.ChatAsync(chatHistory, userMessage);
   chatHistory.Add(("User", userMessage));
   chatHistory.Add(("AI", reply));
}
```

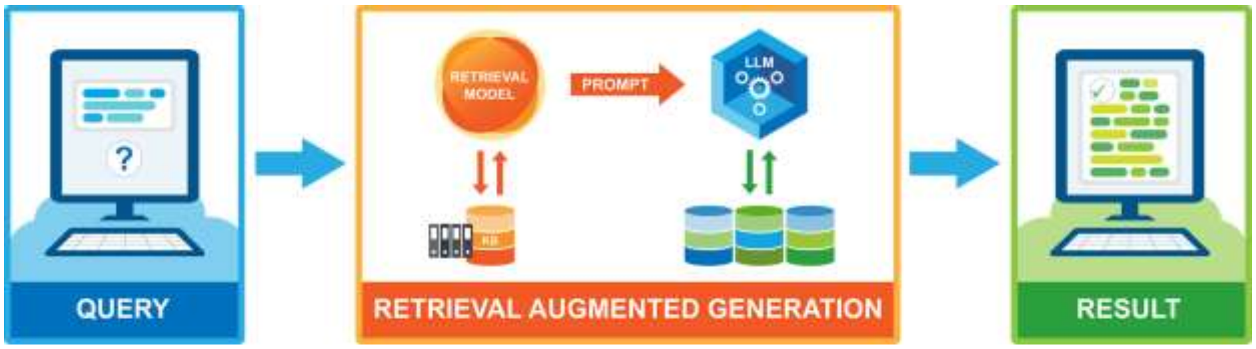
Chat Service Logic

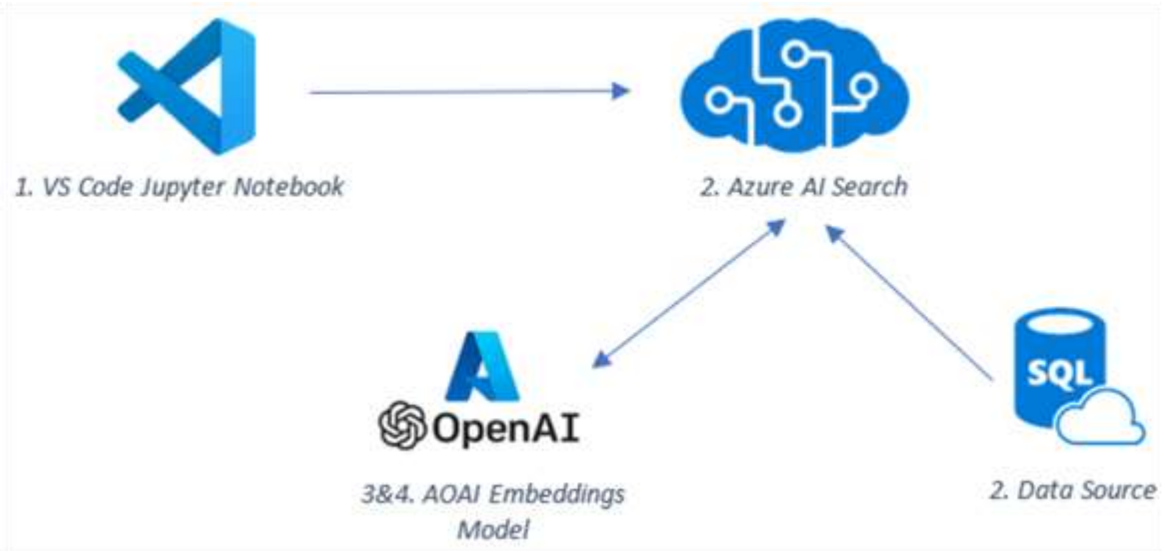
```
public async Task<string> ChatAsync(
   List<(string Role, string Content)> history,
   string userInput)
{
   var messages = history.Select(h =>
      $"{h.Role}: {h.Content}");

   var prompt = string.Join("\n", messages)
      + $"\nUser: {userInput}\nAI:";

   return await SendToOpenAI(prompt);
}
```

💡 **Production tip**

- Persist history in Redis / SQL
- Trim context to avoid token explosion

5️⃣ Connecting Azure OpenAI to an Existing Data Index (RAG)





Why RAG?

LLMs:

- ❌ Do not know your private data
- ❌ Hallucinate

Retrieval-Augmented Generation (RAG) solves this.

High-Level Flow

User Question

↓

Generate Embedding

↓

Search Vector Index (Azure AI Search)

↓

Inject Retrieved Data into Prompt

↓

Azure OpenAI Answer

Embedding Generation

```
public async Task<float[]> CreateEmbedding(string text)
{
   // Call embedding deployment
}
```

Querying Existing Data

```
var results = await _searchClient.SearchAsync<SearchDocument>(
   vector: embedding,
   top: 5
);
```

Final Prompt (RAG)

Answer using ONLY the context below.

If not found, say "Data not available".

Context:

- Product A price is 100

- Product B discontinued

Question:

What is the price of Product A?

🔐 Security & Best Practices

✓ Use **server-side calls only** (never expose API key)
✓ Rate limit AI endpoints
✓ Log prompts & responses (for audits)
✓ Validate user input (prompt injection defense)
✓ Cache responses where possible

🧠 Summary

Feature	**What AI Adds**
Smart Pasting	Clean & structured input
Smart Text Area	Context-aware writing
ChatBot	Conversational UX
Data Index (RAG)	Enterprise-safe AI answers
