Bk

```
from flask import Flask, request, jsonify

from flask_cors import CORS

import openai

import numpy as np


app = Flask(__name__)

CORS(app)


# -------------------------

# Azure OpenAI Configuration

# -------------------------

openai.api_type = "azure"

openai.api_base = "https://hemal-mg3addke-swedencentral.openai.azure.com/"

openai.api_version = "2024-12-01-preview"

openai.api_key =
"70DsK0AZNsgtS6zeRdyeO3lgKX9Cci2mhXpuBPv0ezrPaRpYkOmVJQQJ99BIACfhMk5XJ3w
3AAAAACOGCBsh"


deployment_name = "gpt-35-turbo-demo"     # Chat model deployment name

embedding_model = "embedding-demo"       # Embedding model deployment name


# -------------------------

# Example documents

# -------------------------

documents = [

  {"id": 1, "text": "Angular is a frontend framework by Google."},
```

```
    {"id": 2, "text": "Flask is a lightweight Python web framework."},
    {"id": 3, "text": "Azure OpenAI service allows GPT models in the cloud."}
]

# -------------------------
# Precompute embeddings for documents
# -------------------------
document_embeddings = []
for doc in documents:
    emb = openai.Embedding.create(
        input=doc["text"],
        engine=embedding_model  # Use 'engine' for Azure
    )['data'][0]['embedding']
    document_embeddings.append({"id": doc["id"], "text": doc["text"], "embedding": emb})

# -------------------------
# Cosine similarity function
# -------------------------
def cosine_similarity(a, b):
    a = np.array(a)
    b = np.array(b)
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))

# -------------------------
# API route for frontend
# -------------------------
```

```
@app.route("/ask", methods=["POST"])
def ask():
    data = request.json
    question = data.get("question", "")
    if not question:
        return jsonify({"answer": "No question provided."})

    # Step 1: Get embedding of the question
    q_emb = openai.Embedding.create(
        input=question,
        engine=embedding_model  # Use 'engine' for Azure
    )['data'][0]['embedding']

    # Step 2: Find most similar document
    similarities = [cosine_similarity(q_emb, doc["embedding"]) for doc in
document_embeddings]
    best_index = int(np.argmax(similarities))
    top_doc = document_embeddings[best_index]["text"]

    # Step 3: Ask GPT with the top document as context
    try:
        response = openai.ChatCompletion.create(
            engine=deployment_name,  # Use 'engine' for Azure
            messages=[
                {"role": "system", "content": "You are a helpful assistant."},
                {"role": "user", "content": f"Answer the question based on this document:
{top_doc}\nQuestion: {question}"}
```

```
        ],
        temperature=0.7
      )
      answer = response['choices'][0]['message']['content']
      return jsonify({"answer": answer})

    except Exception as e:
      return jsonify({"answer": f"Error: {str(e)}"})

# -------------------------
# Run server
# -------------------------
if __name__ == "__main__":
    app.run(debug=True)
```

home.ts

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HttpClient, HttpClientModule } from '@angular/common/http';

@Component({
  selector: 'app-home',
  standalone: true,
  imports: [CommonModule, FormsModule, HttpClientModule],
  templateUrl: './home.html',
```

```
  styleUrls: ['./home.css']
})
export class Home {
  prompt: string = '';
  isSubmitting = false;
  submittedPrompts: string[] = [];
  backendResponse: string = '';   // 👈 store backend response

  constructor(private http: HttpClient) {}

  submitPrompt(): void {
    const text = this.prompt.trim();
    if (!text || this.isSubmitting) return;

    this.isSubmitting = true;
    this.backendResponse = '';

    // ✅ Call backend API instead of just simulating
    this.http.post<{ answer: string }>('http://127.0.0.1:5000/ask', { question: text })
      .subscribe({
        next: (res) => {
          console.log('Backend response:', res);
          this.submittedPrompts.push(text);
          this.backendResponse = res.answer;   // store response
          this.prompt = '';
          this.isSubmitting = false;
```

```
      },
      error: (err) => {
        console.error('Error:', err);
        this.backendResponse = ' ⚠️ Could not connect to backend';
        this.isSubmitting = false;
      }
    });
  }

  clearPrompt(): void {
    if (this.isSubmitting) return;
    this.prompt = '';
  }

  get isSubmitDisabled(): boolean {
    return !((this.prompt ?? '').trim()) || this.isSubmitting;
  }
}
```

home.html

```
<div class="home-center dark-theme" style="--header-h: 0px;">
  <section class="prompt-card" role="region" aria-labelledby="promptTitle">
    <h2 class="title" id="promptTitle">Ask anything</h2>

    <label class="sr-only" for="promptInput">Your prompt</label>
    <textarea
```

```
    id="promptInput"
    class="prompt-input"
    [(ngModel)]="prompt"
    placeholder="Type your prompt here..."
    (keydown.control.enter)="submitPrompt()"
    (keydown.meta.enter)="submitPrompt()"
  ></textarea>

  <div class="actions">
    <button class="clear-btn" type="button" (click)="clearPrompt()"
[disabled]="!prompt.length || isSubmitting">
      Clear
    </button>

    <button class="primary-btn" type="button" (click)="submitPrompt()"
[disabled]="isSubmitDisabled">
      {{ isSubmitting ? 'Submitting...' : 'Submit' }}
    </button>
  </div>

  <!-- Submitted prompts displayed below -->
  <div class="submitted-prompts" *ngIf="submittedPrompts.length">
    <h3 class="submitted-title">Submitted Prompts:</h3>
    <div class="prompt-card-small" *ngFor="let p of submittedPrompts">
      {{ p }}
    </div>
  </div>
```

```
    <!-- 👇 Show backend response -->
    <div *ngIf="backendResponse" class="response-box">
      <h3>Backend Response:</h3>
      <p>{{ backendResponse }}</p>
    </div>
  </section>
</div>
```