EmbeddingsService

September 4, 2026

EmbeddingsService

Kind: Service

Source: `atloria-monorepo/apps/api/src/technical-docs/rag/embeddings.service.ts`

Azure OpenAI text embeddings for the technical-docs RAG. Uses the `text-embedding-3-large`
deployment (3072-dim, regional → data stays in-region). Batches inputs per request and retries
transient failures so indexing tens of thousands of entities is resilient.

`EmbeddingsService` generates Azure OpenAI text embeddings for the technical-docs RAG pipeline, using the `text-embedding-3-large` deployment (3072 dimensions) to keep data in-region. It batches inputs to reduce request overhead and implements retries for transient failures, making large-scale indexing (tens of thousands of entities) resilient. This service is typically used during ingestion/indexing and any workflow that needs deterministic vector representations for search and retrieval.

Methods

Method	Signature	Returns	Description
`embedOne`	`embedOne(text: string, usage: { tokens: number })`	`Promise<number[]>`	Embed one string → one vector.
`embed`	`embed(texts: string[], batchSize: unknown, usage: { tokens: number })`	`Promise<number[][]>`	Embed a batch of strings → vectors, in input order.

Dependencies

- `ConfigService`

Where it refuses work

- `EmbeddingsService` stops the work with `Error` when `!this.client` — “Embeddings not configured”.
- `EmbeddingsService` stops the work with an early return when `!texts.length`.

When something fails

- `EmbeddingsService` handles failure in 1 place: it lets it reach the caller in all 1.

Diagram

```
sequenceDiagram
  autonumber
  participant Indexer as RAG Indexer/Ingester
  participant Embeddings as EmbeddingsService
  participant Azure as Azure OpenAI (text-embedding-3-large)
  participant Store as Vector Store/Index

  Indexer->>Embeddings: embedTexts(texts[])
  note over Embeddings: Split into batches\n(avoid max input limits)
  loop For each batch
    Embeddings->>Azure: createEmbeddings(batch)
    alt transient failure (429/5xx/network)
      Embeddings->>Azure: retry with backoff
    end
    Azure-->>Embeddings: embeddings[]
  end
  Embeddings-->>Indexer: embeddings aligned to inputs
  Indexer->>Store: upsert(vectors + metadata)
```

Usage

```
import { Injectable } from '@nestjs/common';
import { EmbeddingsService } from './embeddings.service';

@Injectable()
export class DocsIndexer {
  constructor(private readonly embeddings: EmbeddingsService) {}

  async indexEntities(entities: Array<{ id: string; title: string; body: string }>) {
    const texts = entities.map(e => `${e.title}\n\n${e.body}`);

    // Produces 3072-dim vectors (text-embedding-3-large)
    const vectors = await this.embeddings.embedTexts(texts);

    // Example: align embeddings back to entities for vector-store upsert
    const records = entities.map((e, i) => ({
```

```
      id: e.id,
      vector: vectors[i],
      metadata: { title: e.title },
    }));

    // upsertRecords(records) // your vector store integration
    return records;
  }
}
```

AI Coding Instructions

- Keep embeddings aligned with input order: batch splitting must preserve stable indexing so `vectors[i]` maps to `texts[i]`.
- Follow the existing batching strategy (size and token/length constraints) to avoid Azure OpenAI input limits and reduce request overhead.
- Retain and extend the retry policy only for transient errors (e.g., 429, 5xx, network timeouts); avoid retrying deterministic validation errors.
- Integration point: this service should be called from ingestion/indexing paths; store the resulting vectors with the same entity IDs/metadata used by the retriever.
- Treat all content as region-bound: do not add logging or telemetry that leaks raw text outside the intended region or persistence boundaries.

Relationships

- DEPENDS_ON → `configservice`

Referenced By

- `TechDocsRagService` (DEPENDS_ON)
- `TechnicalDocsModule` (MODULE_PROVIDES)