TechDocsRagService

September 4, 2026

TechDocsRagService

Kind: Service

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

Grounded codebase chat (GraphRAG) over the technical docs.

index(projectId): embed each published page → upsert as (:TechChunk) with its vector in Neo4j,
backed by a native vector index.
ask(projectId, q): embed the question → Neo4j vector search (scoped to the project) → feed the
top hits to gpt-5.x with citation-forced grounding → return an answer + validated citations
(invented references are dropped by validateCitations).

The same ask() backs both the web "ask about this codebase" endpoint and the MCP ask_question tool.

`TechDocsRagService` provides grounded, citation-backed Q&A over a project’s published technical documentation using a GraphRAG-style pipeline. It indexes docs by embedding each published page into vectorized `(:TechChunk)` nodes in Neo4j (with a native vector index), then answers questions by running scoped vector search and prompting `gpt-5.x` with strict citation grounding. It powers both the web “ask about this codebase” endpoint and the MCP `ask_question` tool, dropping any invented references via `validateCitations`.

Methods

Method	Signature	Returns	Description
`index`	`index(projectId: string)`	`Promise<{ indexed: number;`	Embed every published page of a project's

Method	Signature	Returns	Description
		`embeddingTokens: number }>`	active technical set and upsert vectors into Neo4j.
`restampVisibility`	`restampVisibility(projectId: string, paths: string[], precomputedKeys: Map<string, string>)`	`Promise<{ stamped: number }>`	D4: update `visibilityKey` on a project's existing chunks WITHOUT re-embedding — a rule change is pure metadata (content is untouched).
`retrieve`	`retrieve(projectId: string, question: string, k: unknown, docTypes: Array<'technical'	'user_manual'	'external'>, visibleKeys: string[]
`purgePath`	`purgePath(projectId: string, pathPrefix: string)`	`Promise<void>`	Remove every trace of a project from the vector store: chunks (batched) + its per-tenant vector index.
`purgeProject`	`purgeProject(projectId: string)`	`Promise<void>`	
`visibleKeysFor`	`visibleKeysFor(projectId: string, attrs: Record<string, unknown>`	null)`	`Promise<string[]
`askFn`	`askFn(projectId: string, visibleKeys: string[]	null)`	`AskFn`
`ask`	`ask(projectId: string, question: string, visibleKeys: string[]	null)`	`Promise<{

```
answer: string;
citations: string[];
sources: DocSearchHit[];
confidence: RagConfidence;
usage: { inputTokens: number; outputTokens: number; totalTokens: number };
```

}> | One-shot: retrieve + grounded answer for a project (with token usage for metering). | | assistPage | assistPage(projectId: string, currentMarkdown: string, instruction: string, pageTitle: unknown) | Promise<{
content: string;
citations: string[];
sources: DocSearchHit[];
usage: { inputTokens: number; outputTokens: number; totalTokens: number };
}>` | Grounded AI authoring: enhance/rewrite/expand a specific documentation page per an instruction, staying grounded in the page's own verified facts (the page i… |

Dependencies

- `Neo4jService`
- `EmbeddingsService`
- `RagChatModelService`
- `TechnicalDocsMaterializerService`
- `DocsVisibilityService`
- `PrismaService`

Where it refuses work

- `TechDocsRagService` stops the work with `Error` when `!this.embeddings.available` — “Embeddings not configured”.
- `TechDocsRagService` stops the work with `Error` when `!projectId` — “retrieve() requires a projectId (tenant scope)”.
- `TechDocsRagService` stops the work with an early return when `!hits.length`, in 2 places.
- `TechDocsRagService` stops the work with an early return when `!variants.length`.
- `TechDocsRagService` stops the work with an early return when `this.indexEnsured`.
- `TechDocsRagService` stops the work with an early return when `!corpus || !corpus.pages.length`.

When something fails

- `TechDocsRagService` handles failure in 2 places: it turns it into a return value in 1, and discards it silently in 1. A failure discarded silently leaves no trace for whoever debugs this later.

Diagram

```
sequenceDiagram
  autonumber
  actor User
  participant API as API Endpoint / MCP Tool
  participant Svc as TechDocsRagService
  participant Embed as Embeddings Provider
  participant Neo4j as Neo4j (Vector Index)
  participant LLM as gpt-5.x
  participant Val as validateCitations

  rect rgb(245,245,245)
    note over Svc,Neo4j: Indexing: index(projectId)
    Svc->>API: index(projectId) triggered
    Svc->>Embed: Embed each published page
    Embed-->>Svc: Page vectors
    Svc->>Neo4j: Upsert (:TechChunk) + vector<br/>scoped to projectId
    Neo4j-->>Svc: Indexed
  end

  rect rgb(245,245,245)
    note over User,Val: Q&A: ask(projectId, q)
    User->>API: Ask question (q)
    API->>Svc: ask(projectId, q)
    Svc->>Embed: Embed question
    Embed-->>Svc: Question vector
    Svc->>Neo4j: Vector search (projectId scope)
    Neo4j-->>Svc: Top-k matching chunks (+ metadata)
    Svc->>LLM: Prompt with chunks + citation-forced grounding
    LLM-->>Svc: Draft answer + citations
    Svc->>Val: Validate citations (drop invented refs)
    Val-->>Svc: Answer + validated citations
    Svc-->>API: Response
    API-->>User: Grounded answer
  end
```

Usage

```
// NestJS-style usage (e.g., in a controller or another service)
import { Controller, Post, Param, Body } from '@nestjs/common';
import { TechDocsRagService } from './technical-docs/rag/techdocs-rag.service';

@Controller('projects/:projectId/techdocs')
export class TechDocsController {
  constructor(private readonly techDocsRag: TechDocsRagService) {}

  @Post('index')
  async index(@Param('projectId') projectId: string) {
```

```
    // Embeds each published page and upserts (:TechChunk) with vectors into Neo4j
    await this.techDocsRag.index(projectId);
    return { ok: true };
  }

  @Post('ask')
  async ask(
    @Param('projectId') projectId: string,
    @Body() body: { q: string },
  ) {
    // Vector-searches within the project scope and returns an answer + validated citations
    const result = await this.techDocsRag.ask(projectId, body.q);
    return result; // typically: { answer: string, citations: [...] }
  }
}
```

AI Coding Instructions

- Keep `ask(projectId, q)` strictly grounded: only feed Neo4j-retrieved chunks to the model and run `validateCitations` to drop any invented references.
- Always scope retrieval and indexing by `projectId` (both Neo4j upserts and vector searches) to prevent cross-project leakage.
- Treat `index(projectId)` as idempotent: upsert `(:TechChunk)` deterministically so re-indexing doesn't duplicate or drift.
- When modifying prompts or chunk metadata, preserve the citation contract (stable identifiers/URLs/anchors) so validation can reliably match citations to retrieved chunks.
- Watch for embedding/vector index mismatches (dimension/model changes): ensure Neo4j vector index configuration stays aligned with the embedding provider used by both `index` and `ask`.

Relationships

- DEPENDS_ON → `Neo4jService`
- DEPENDS_ON → `EmbeddingsService`
- DEPENDS_ON → `RagChatModelService`
- DEPENDS_ON → `TechnicalDocsMaterializerService`
- DEPENDS_ON → `DocsVisibilityService`
- DEPENDS_ON → `PrismaService`

Referenced By

- `ManualsInsightsService` (DEPENDS_ON)
- `ContentVisibilityService` (DEPENDS_ON)
- `ProjectService` (DEPENDS_ON)
- `PublicProjectController` (DEPENDS_ON)
- `TechDocsChatService` (DEPENDS_ON)
- `TechDocsSourcesService` (DEPENDS_ON)
- `TechnicalDocsMcpController` (DEPENDS_ON)
- `TechnicalDocsQueue` (DEPENDS_ON)
- `TechnicalDocsModule` (MODULE_PROVIDES)
- `TechnicalDocsModule` (MODULE_EXPORTS)