TechnicalDocsParserService

September 4, 2026

TechnicalDocsParserService

Kind: Service

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

Parses a git repository and builds a TechnicalSnapshot:

- entity graph (Entity[] + Relationship[])
- navMap (3-level: apps/libs → type → entities)
- OpenAPI spec (from API_ENDPOINT entities)

Also imports standalone repo MD files as Documents linked via kgEntityId,
and indexes all entities into Azure AI Search as technical_ref.

`TechnicalDocsParserService` scans a git repository to produce a `TechnicalSnapshot` used by the platform's technical documentation and knowledge graph features. It builds the entity graph (entities + relationships), a 3-level navigation map (apps/libs → type → entities), and derives an OpenAPI spec from `API_ENDPOINT` entities. It also ingests standalone Markdown files as `Document` nodes linked via `kgEntityId`, and indexes all parsed entities into Azure AI Search under the `technical_ref` index.

Methods

Method	Signature	Returns	Description
`run`	`run(options: TechnicalDocsJobOptions)`	`Promise<TechnicalDocsJobResult>`	Run the full technical docs parse pipeline for a project.

Dependencies

- `PrismaService`
- `GitCloneService`

- `TechnicalDocsService`
- `DocumentIndexingService`
- `TechnicalDocsEnrichmentService`

Where it refuses work

- `TechnicalDocsParserService` stops the work with `Error` when `!cloneResult.success`.
- `TechnicalDocsParserService` stops the work with an early return when `!decorators.length`.
- `TechnicalDocsParserService` stops the work with an early return when `/@(Public|AllowAnonymous|SkipAuth|NoAuth)\b/.test(all)`.
- `TechnicalDocsParserService` stops the work with an early return when `/@ApiBearerAuth|@ApiSecurity/.test(all)`.
- `TechnicalDocsParserService` stops the work with an early return when `!guards.length`.
- `TechnicalDocsParserService` stops the work with an early return when `guards.some((g) => /^Optional/i.test(g))`.

When something fails

- `TechnicalDocsParserService` handles failure in 8 places: it logs it and continues in 6, turns it into a return value in 1, and lets it reach the caller in 1.

Diagram

```
sequenceDiagram
  autonumber
  participant Caller
  participant Parser as TechnicalDocsParserService
  participant Repo as Git Repo (apps/libs)
  participant MD as Standalone MD Files
  participant KG as TechnicalSnapshot (graph+navMap+openapi)
  participant Search as Azure AI Search (technical_ref)

  Caller->>Parser: parseRepository(repoPath | gitRef)
  Parser->>Repo: traverse source tree
  Repo-->>Parser: code entities (apps/libs/types)
  Parser->>Parser: build Entity[] + Relationship[]
  Parser->>Parser: build navMap (apps/libs → type → entities)
  Parser->>Parser: derive OpenAPI spec from API_ENDPOINT entities

  Parser->>MD: scan/import *.md
  MD-->>Parser: Documents (linked via kgEntityId)
```

```
  Parser-->>KG: TechnicalSnapshot
  Parser->>Search: index entities as technical_ref
  Search-->>Parser: indexing result
  Parser-->>Caller: return TechnicalSnapshot
```

Usage

```
// Example in a NestJS context (e.g., inside a controller or another service)
import { Injectable } from '@nestjs/common';
import { TechnicalDocsParserService } from './technical-docs-parser.service';

@Injectable()
export class TechnicalDocsJob {
  constructor(private readonly parser: TechnicalDocsParserService) {}

  async run() {
    // Adjust inputs to match your app's configuration (repo path, git ref, tenant/project id,
    const snapshot = await this.parser.parseRepository({
      repoPath: '/var/repos/atloria-monorepo',
      // gitRef: 'main',
      // includeDocs: true,
      // indexToSearch: true,
    });

    // Use snapshot outputs downstream
    console.log('Entities:', snapshot.entities.length);
    console.log('Relationships:', snapshot.relationships.length);
    console.log('NavMap keys:', Object.keys(snapshot.navMap));
    console.log('OpenAPI paths:', Object.keys(snapshot.openApi?.paths ?? {}));
  }
}
```

AI Coding Instructions

- Keep parsing deterministic: stable entity IDs, consistent relationship directions, and sorted outputs to avoid noisy diffs and re-index churn.
- Treat `API_ENDPOINT` entities as the single source of truth for OpenAPI generation; validate required fields early to prevent producing partial specs.
- When importing standalone Markdown, always link to the KG via `kgEntityId` and avoid creating duplicate `Document` nodes for the same target.
- Azure AI Search integration should be idempotent and resilient: batch indexing, handle retries, and ensure the `technical_ref` schema stays compatible with entity shape changes.

- Maintain the 3-level `navMap` contract (apps/libs → type → entities); changing its shape will break consumers in docs UI and graph explorers.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `GitCloneService`
- DEPENDS_ON → `TechnicalDocsService`
- DEPENDS_ON → `DocumentIndexingService`
- DEPENDS_ON → `TechnicalDocsEnrichmentService`

Referenced By

- `TechnicalDocsGenerationService` (DEPENDS_ON)
- `TechnicalDocsModule` (MODULE_PROVIDES)
- `TechnicalDocsModule` (MODULE_EXPORTS)