DocsRepoService

September 4, 2026

DocsRepoService

Kind: Service

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

DocsRepoService — the low-level git primitive for the per-project docs substrate.

One NON-BARE git repo per project lives at `${DOCS_REPO_ROOT}/<projectId>` (default /work/docs-repos, overridable for tests). Every mutation goes through git so the repo IS the source of record for content history; Postgres remains the authority the repo is rebuildable from (see ReconcilerService).

ALL git invocation is via promisified execFile('git', [args], { cwd }) — an argv array, never a shell string — so a slug/path/message can never inject a shell command.

`DocsRepoService` manages the non-bare Git repository that backs each project's documentation content under `${DOCS_REPO_ROOT}/<projectId>`. It safely initializes repositories, normalizes and writes document trees, and exposes Git-derived state such as tree hashes and normalizer versions; higher-level reconciliation can rebuild repository state from Postgres when needed.

Methods

Method	Signature	Returns	Description
`normalizerVersionTag`	`normalizerVersionTag()`	`string`	The normalizer version this service stamps on commits (see driftCheck).
`repoRoot`	`repoRoot()`	`string`	Root under which every

Method	Signature	Returns	Description
			project's repo lives.
`repoPath`	`repoPath(projectId: string)`	`string`	Absolute path to a project's working tree.
`normalizeContent`	`normalizeContent(markdown: string)`	`string`	Expose the canonicaliser so callers (drift compare) normalise DB content identically.
`repoRelPath`	`repoRelPath(p: string)`	`string`	The EXACT git-normalised repo path a write would use (safe-relative + forward slashes).
`exists`	`exists(projectId: string)`	`Promise<boolean>`	True once `git init` has run for this project.
`ensureRepo`	`ensureRepo(projectId: string)`	`Promise<void>`	Idempotent: create the repo if absent (git init on `main`, typed generator identity, one empty root commit so the tree is never in an unborn-HEAD state).
`writeFiles`	`writeFiles(projectId: string, files: RepoFile[], author: CommitAuthor, message: string,`	`Promise<string>`	Normalise + write each file, stage it, and commit

Method	Signature	Returns	Description
	`branch: string, opts: CommitOptions)`		with the TYPED author.
`writeTree`	`writeTree(projectId: string, files: RepoFile[], author: CommitAuthor, message: string, branch: string, sidecar: RepoFile[], opts: CommitOptions, snippetFiles: RepoFile[])`	`Promise<string>`	FULL-TREE projection under docs/ (the reconciler's rebuild primitive).
`headNormalizerVersion`	`headNormalizerVersion(projectId: string)`	`Promise<string | null>`	
`readFile`	`readFile(projectId: string, filePath: string, ref: unknown)`	`Promise<string>`	Read a file's content at `ref` (default HEAD) via `git show`.
`resolveSlugAlias`	`resolveSlugAlias(projectId: string, oldSlug: string, scope: string)`	`Promise<string | null>`	
`listFiles`	`listFiles(projectId: string, ref: unknown)`	`Promise<string[]>`	All tracked file paths at `ref` (default HEAD).
`headCommit`	`headCommit(projectId: string)`	`Promise<string | null>`	
`checkout`	`checkout(projectId: string, ref: string)`	`Promise<void>`	Check out `ref`; if it's an unknown branch name, create it from current HEAD.
`checkoutForce`	`checkoutForce(projectId: string, ref: string)`	`Promise<void>`	FORCE check out an EXISTING ref, discarding any uncommitted

Method	Signature	Returns	Description
			working-tree changes.
`createBranch`	`createBranch(projectId: string, name: string, fromRef: string)`	`Promise<void>`	Create a branch (from `fromRef` or current HEAD) without switching to it.
`merge`	`merge(projectId: string, branch: string, message: string)`	`Promise<void>`	Merge `branch` into the current branch (fast-forward or a merge commit).
`tag`	`tag(projectId: string, name: string, ref: string, message: string)`	`Promise<void>`	Create a tag (annotated when a message is given) at `ref` or current HEAD.
`commitDetachedTree`	`commitDetachedTree(projectId: string, files: RepoFile[], tagName: string, message: string)`	`Promise<string>`	Create a git TAG whose commit captures `files` as a full `docs/` tree, WITHOUT touching HEAD, the working tree, or the repo index — via a throwaway GIT_INDEX…

Dependencies

- `ConfigService`
- `NormalizeFn`

Where it refuses work

- `DocsRepoService` stops the work with `Error` when `!gitPath.startsWith('snippets/')`.
- `DocsRepoService` stops the work with `Error` when `!gitPath.startsWith('.atloria/')`.
- `DocsRepoService` stops the work with `Error` when `!seg || seg.includes('/') || seg.includes('\\') || seg === '.' || seg === '..'`.
- `DocsRepoService` stops the work with `Error` when `!p` — “Empty file path”.
- `DocsRepoService` stops the work with `Error` when `path.isAbsolute(rel) || rel.split(path.sep).some((s) => s === '..')`.
- `DocsRepoService` stops the work with an early return when `!(await this.hasStagedChanges(cwd))`, in 2 places.

When something fails

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

Diagram

```
sequenceDiagram
  participant Caller as Reconciler / Docs Service
  participant Repo as DocsRepoService
  participant FS as Project Docs Repository
  participant Git as git (execFile)

  Caller->>Repo: ensureRepo(projectId)
  Repo->>FS: Check repository path
  alt Repository does not exist
    Repo->>Git: git init (argv array)
    Git-->>Repo: Repository initialized
  end
  Repo-->>Caller: Ready

  Caller->>Repo: writeTree(projectId, files)
  Repo->>Repo: normalizeContent(content)
  Repo->>Repo: repoRelPath(path)
  Repo->>FS: Write normalized files
  Repo->>Git: git add / write-tree
  Git-->>Repo: tree SHA
  Repo-->>Caller: tree SHA
```

Usage

```
import { DocsRepoService } from './docs-repo.service';

// Typically injected by NestJS into a reconciler or document mutation service.
export class ProjectDocsWriter {
  constructor(private readonly docsRepoService: DocsRepoService) {}

  async writeProjectDocs(projectId: string) {
    await this.docsRepoService.ensureRepo(projectId);

    const treeSha = await this.docsRepoService.writeTree(projectId, [
      {
        path: 'README.md',
        content: '# Project Documentation\n\nWelcome to the project docs.\n',
      },
      {
        path: 'guides/getting-started.md',
        content: '# Getting Started\n\nInstall the application and configure it.\n',
      },
    ]);

    return { projectId, treeSha };
  }
}
```

AI Coding Instructions

- Route all repository mutations through `DocsRepoService`; do not write directly to project documentation repositories from callers.
- Preserve the `execFile('git', args, { cwd })` pattern for every Git command. Never construct shell command strings from project IDs, paths, or commit messages.
- Validate and convert document paths with `repoRelPath()` before file operations to prevent writes outside the project repository.
- Normalize content with `normalizeContent()` before writing so generated trees are deterministic and compatible with normalizer-version checks.
- Treat Git as the content-history record, while using reconciliation integrations to rebuild repository content from Postgres when repository state is missing or stale.

Relationships

- DEPENDS_ON → `configservice`
- DEPENDS_ON → `NormalizeFn`

Referenced By

- `DiffService` (DEPENDS_ON)
- `DocsRepoModule` (MODULE_PROVIDES)
- `DocsRepoModule` (MODULE_EXPORTS)
- `ReconcilerService` (DEPENDS_ON)
- `GitSyncMirrorService` (DEPENDS_ON)
- `PublicProjectController` (DEPENDS_ON)