TechnicalDocsService

September 4, 2026

TechnicalDocsService

Kind: Service

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

`TechnicalDocsService` manages project-scoped technical documentation settings and generated documentation artifacts for the API. It resolves the project context for jobs, retrieves snapshots and API documentation sets, and manages curation, playground test-token, escalation, and API target configuration.

Methods

Method	Signature	Returns	Description
`resolveProjectForJob`	`resolveProjectForJob(projectId: string, user: JwtPayload, opts: { requireRepo?: boolean })`	`unknown`	Resolve project for a jc validates access and (I ensures repoUrl is set.
`getSnapshot`	`getSnapshot(projectId: string)`	`unknown`	Get the technical snap project (entities + nav r
`getApiDocSet`	`getApiDocSet(projectId: string, userId: string)`	`unknown`	API Reference doc-set (docKind:'api').
`setCuration`	`setCuration(projectId: string, userId: string, dto: { pinned?: unknown; hidden?: unknown })`	`Promise<TechdocsCuration>`	Persist curation (pinne entity ids) into project.settings.techdo
`setPlaygroundTestToken`	`setPlaygroundTestToken(projectId: string, userId: string, token: unknown)`	`Promise<{ hasTestToken: boolean }>`	Playground test key (P a maintainer-provided credential that pre-fills playground's Authorize "try it" works without hu
`getPlaygroundTestToken`	`getPlaygroundTestToken(projectId: string, userId: string)`	`Promise<{ token: string`	null; testTokenPublic: b
`setPlaygroundTestTokenVisibility`	`setPlaygroundTestTokenVisibility(projectId: string, userId: string, publicVisible: boolean)`	`Promise<{ testTokenPublic: boolean }>`	Opt-in: whether the tes fills the PUBLIC docs p too (C1).
`setEscalationConfig`	`setEscalationConfig(projectId: string, userId: string, dto: { webhookUrl?: string })`	`unknown`	Escalation config (3.1): "escalate to a human" — a Slack-compatible URL.
`getEscalationConfig`	`getEscalationConfig(projectId: string, userId: string)`	`unknown`	
`getApiTargetConfig`	`getApiTargetConfig(projectId: string, userId: string)`	`Promise<ApiTargetPublicConfig>`	Live API target (docs-a executable-MCP-tools) customer's opt-in confi that lets an MCP agent their real API via `call_operation`.
`setApiTargetConfig`	`setApiTargetConfig(projectId: string, userId: string, dto: {		

```
  baseUrl?: string;
  authHeaderName?: string;
  authHeaderValue?: string;
  allowWrites?: boolean;
  enabled?: boolean;
  writeAllowlist?: string[];
```

```
  oauth2?: PlaygroundOAuth2Input | null;
})` | `Promise<ApiTargetPublicConfig>` | Merge an API-target update into project.settings.apiExec. |
```

`getApiExecRuntimeConfig`	`getApiExecRuntimeConfig(projectId: string)`	`Promise<ApiExecRuntimeConfig	null>`	Runtime config used to BUILD the executor (includes the secret auth value).
`getEntity`	`getEntity(projectId: string, entityId: string)`	`unknown`	Get a single entity with its linked Documents (user MD files + AI enrichments)
`getApiSpec`	`getApiSpec(projectId: string)`	`unknown`	Get the OpenAPI spec for the API playground
`getJobStatus`	`getJobStatus(projectId: string)`	`unknown`	Get job status for a running technical docs generation
`upsertSnapshot`	`upsertSnapshot(projectId: string, data: { entitiesJson: any[]; relationshipsJson: any[]; apiSpecJson?: any; navMapJson: any; // Baseline for incremental staleness: the commit/branch these docs were generated from. generatedFromCommit?: string	null; generatedFromBranch?: string	null; })`	`unknown`	Upsert a technical snapshot (called by the parser pipeline)
`checkStaleness`	`checkStaleness(projectId: string, changedFiles: string[], commit: string, branch: string)`	`Promise<{ staleCount: number }	null>`	Freshness notify-mode (Tier 2c): map the files changed in a push onto the entity graph and record how many published pages are now stale.
`getStaleEntityIds`	`getStaleEntityIds(projectId: string)`	`Promise<string[]>`	The FULL stale-entity id set for a project.
`clearStaleness`	`clearStaleness(projectId: string)`	`Promise<void>`	Clear the stale flag — called after a successful regeneration publishes fresh docs.
`createTrackedJob`	`createTrackedJob(projectId: string, organizationId: string, userId: string, stage: string, branch: string | null)`	`Promise<string	null>`	Create a DocumentationJob so EVERY pipeline run is tracked (status/stage/duration/error) — previously jobs ran untracked unless the project happened to have …

Dependencies

- `PrismaService`
- `DocumentIndexingService`
- `StaleAlertService`
- `SdkArtifactsService`

Where it refuses work

- `TechnicalDocsService` stops the work with `NotFoundException` when `!project` — “Project not found”, in 10 places.
- `TechnicalDocsService` stops the work with `NotFoundException` when `!snapshot` — “No technical snapshot found for this project.”, in 2 places.
- `TechnicalDocsService` stops the work with `BadRequestException` when `requireRepo && !project.repoUrl` — “Project has no repository URL configured. Set it in project settings first.”.
- `TechnicalDocsService` stops the work with `NotFoundException` when `!snapshot` — “No technical snapshot found for this project. Generate technical docs first.”.
- `TechnicalDocsService` stops the work with `BadRequestException` when `typeof token !== 'string' || token.length > 4096` — “token must be a string (max 4096 chars); empty string clears it”.
- `TechnicalDocsService` stops the work with `BadRequestException` when `webhookUrl.length > 2048` — “webhookUrl too long”.

When something fails

- `TechnicalDocsService` handles failure in 2 places: it logs it and continues in 1, and turns it into a return value in 1.

Diagram

```
sequenceDiagram
  participant Client
  participant Controller
  participant Service as TechnicalDocsService
  participant Project as Project/Job Context
  participant Store as Persistence/Config Store
```

```
Client->>Controller: Request technical docs configuration
Controller->>Service: resolveProjectForJob()
Service->>Project: Resolve project from job context
Project-->>Service: Project context

alt Read documentation data
  Controller->>Service: getSnapshot() / getApiDocSet()
  Service->>Store: Load generated docs artifacts
  Store-->>Service: Snapshot or API doc set
  Service-->>Controller: Documentation data
else Update configuration
  Controller->>Service: setCuration() / setEscalationConfig()
  Service->>Store: Persist project-scoped configuration
  Store-->>Service: Updated configuration
  Service-->>Controller: Configuration result
else Manage playground token
  Controller->>Service: setPlaygroundTestToken(...)
  Service->>Store: Store token and visibility
  Store-->>Service: Token state
  Service-->>Controller: Public token metadata
end
```

Usage

```
import { Injectable } from '@nestjs/common';
import { TechnicalDocsService } from './technical-docs.service';

@Injectable()
export class DocsAdminService {
  constructor(
    private readonly technicalDocsService: TechnicalDocsService,
  ) {}

  async configurePlaygroundToken(token: string) {
    // Token values should only be obtained through authorized admin flows.
    const result = await this.technicalDocsService.setPlaygroundTestToken(
      token,
    );

    await this.technicalDocsService.setPlaygroundTestTokenVisibility(true);

    return {
      hasTestToken: result.hasTestToken,
      testTokenPublic: true,
    };
  }

  async getDocumentationOverview() {
    const [snapshot, apiDocSet, targetConfig] = await Promise.all([
      this.technicalDocsService.getSnapshot(),
      this.technicalDocsService.getApiDocSet(),
      this.technicalDocsService.getApiTargetConfig(),
    ]);

    return { snapshot, apiDocSet, targetConfig };
  }
}
```

AI Coding Instructions

- Resolve the project/job context before reading or updating project-scoped technical documentation data.
- Keep playground test tokens confidential; use `getPlaygroundTestToken()` only in authorized server-side flows and avoid logging its `token` value.
- Use `setPlaygroundTestTokenVisibility()` independently when changing whether a configured token is exposed to the playground.
- Preserve the public response contracts for token and API target methods, especially `hasTestToken`, `testTokenPublic`, and `ApiTargetPublicConfig`.
- Integrate curation and escalation updates through this service rather than writing documentation configuration directly from controllers or jobs.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `DocumentIndexingService`
- DEPENDS_ON → `StaleAlertService`
- DEPENDS_ON → `SdkArtifactsService`

Referenced By

- `PublicProjectController` (DEPENDS_ON)
- `TechnicalDocsChatController` (DEPENDS_ON)
- `TechnicalDocsGenerationService` (DEPENDS_ON)
- `TechnicalDocsMcpController` (DEPENDS_ON)
- `TechnicalDocsParserService` (DEPENDS_ON)
- `TechnicalDocsController` (DEPENDS_ON)
- `TechnicalDocsModule` (MODULE_PROVIDES)
- `TechnicalDocsModule` (MODULE_EXPORTS)
- `TrialService` (DEPENDS_ON)
- `WebhookService` (DEPENDS_ON)