# DocVersionService

September 4, 2026

## DocVersionService

**Kind:** Service

**Source:** `atloria-monorepo/apps/api/src/doc-version/doc-version.service.ts`

`DocVersionService` manages document version lifecycle operations for projects, including version numbering, retrieval, activation, publishing, and scheduled activation. It also coordinates RAG reindexing when version state changes so downstream AI search and retrieval systems use the appropriate document content.

### Methods

| Method | Signature | Returns | Description |
| --- | --- | --- | --- |
| `getNextVersionNumber` | `getNextVersionNumber(projectId: string)` | `Promise<number>` | Get the next version number for a project (auto-increment). |
| `create` | `create(projectId: string, organizationId: string, createdById: string, opts: {` | | |

```
  label?: string;
  description?: string;
  commitHash?: string;
  branch?: string;
  gitTag?: string;
  previousVersionId?: string;
  jobId?: string;
  /** Doc set this version belongs to: 'user' (default) or 'technical'. */
  audienceScope?: string;
})` | `unknown` | Create a new DocVersion (typically called at start of generation). |
```

| `listByProject` | `listByProject(projectId: string, opts: { status?: DocVersionStatus[]; limit?: number; offset?: number; })` | `unknown` | List all versions for a project with document/category counts. |
| `getById` | `getById(id: string)` | `unknown` | Get a specific version by ID with counts. |
| `getActiveVersion` | `getActiveVersion(projectId: string, audienceScope: unknown)` | `unknown` | Get the active version for a project (status = ACTIVE) within one audience scope. |
| `getPublishedVersions` | `getPublishedVersions(projectId: string)` | `unknown` | Get all published/accessible versions for version switcher. |
| `activate` | `activate(id: string)` | `unknown` | Activate a version — sets it to ACTIVE, demotes the previous ACTIVE to PUBLISHED. |
| `scheduleActivation` | `scheduleActivation(versionId: string, projectId: string, opts: { at: Date; timezone?: string }, userId: string)` | `unknown` | D5: Schedule a version to go live (activate) at a future time. |
| `cancelSchedule` | `cancelSchedule(versionId: string, projectId: string, userId: string)` | `unknown` | D5: Cancel a pending scheduled activation. |
| `enqueueRagReindex` | `enqueueRagReindex(projectId: string, organizationId: string, userId: string \| null)` | `void` | Enqueue a RAG re-index (fast lane) of a project's published corpus. |
| `publish` | `publish(id: string)` | `unknown` | Publish a version (multi-version publishing, not the primary active one). |
| `archive` | `archive(id: string)` | `unknown` | Archive a version (read-only, kept for history). |
| `deprecate` | `deprecate(id: string)` | `unknown` | Deprecate a version (still accessible, shows deprecation banner). |
| `update` | `update(id: string, dto: { label?: string; description?: string })` | `unknown` | Update version metadata (label, description). |
| `delete` | `delete(id: string)` | `unknown` | Delete a version (only DRAFT versions can be deleted). |
| `setGenerationSummary` | `setGenerationSummary(id: string, summary: Record<string, any>)` | `unknown` | Set generation summary after job completes. |
| `getReviewData` | `getReviewData(versionId: string)` | `unknown` | Phase 2: Get review data for a version (pre-publish review). |
| `recordFlagResolution` | `recordFlagResolution(versionId: string, previousDocId: string)` | `Promise<void>` | Regen→CR interlock (S2B): record that a flagged carry-forward conflict (flag id = the flagged doc's previousDocId) was resolved on the /review screen, then a… |
| `getScreenshotInventory` | `getScreenshotInventory(versionId: string)` | `unknown` | Phase 4: Get screenshot inventory for a version. |

| `saveScreenshotAnnotations` | `saveScreenshotAnnotations(projectId: string, placeholderId: string, raw: unknown)` | `unknown` | Persist authored annotations for one screenshot placeholder (arrow/box/mask/marker overlay). |
| `removeScreenshotPlaceholders` | `removeScreenshotPlaceholders(versionId: string, routePaths: string[])` | `unknown` | Phase 4: Bulk screenshot action — remove placeholders from document content. |
| `previewScreenshot` | `previewScreenshot(url: string)` | `unknown` | Phase 4: Preview a screenshot — capture with Playwright and return as base64 (don't save). |
| `captureScreenshot` | `captureScreenshot(projectId: string, versionId: string, routePath: string, documentId: string, userTargetUrl: string)` | `unknown` | Phase 4: Capture a screenshot using Playwright and save it. |

## Dependencies

- `PrismaService`
- `AuditService`
- `ModuleRef` *(optional)*
- `VersionIndexEvictor` *(optional)*
- `DocVersionExportService` *(optional)*
- `ChangelogDraftsService` *(optional)*

## Where it refuses work

- `DocVersionService` stops the work with `NotFoundException` when `version.projectId !== projectId` — “DocVersion not found in this project”, in 2 places.
- `DocVersionService` stops the work with `BadRequestException` when `!allowed.includes(to)`.
- `DocVersionService` stops the work with `NotFoundException` when `!version` — “DocVersion not found”.
- `DocVersionService` stops the work with `BadRequestException` when `!approval` — “This version requires approval before publishing. Request approval first.”.
- `DocVersionService` stops the work with `BadRequestException` when `!(opts.at instanceof Date) || Number.isNaN(opts.at.getTime())` — “Invalid schedule time — provide a valid ISO date-time.”.
- `DocVersionService` stops the work with `BadRequestException` when `opts.at.getTime() < Date.now() + MIN_LEAD_MS` — “Scheduled time must be at least 2 minutes in the future.”.

## When something fails

- `DocVersionService` handles failure in 6 places: it lets it reach the caller in 3, turns it into a return value in 2, and discards it silently in 1. A failure discarded silently leaves no trace for whoever debugs this later.

## Diagram

```
sequenceDiagram
  participant Client
  participant Controller
  participant Service as DocVersionService
  participant Store as Version Persistence
  participant Queue as RAG Reindex Queue

  Client->>Controller: Create or activate document version
  Controller->>Service: create() / activate()
  Service->>Service: getNextVersionNumber()
  Service->>Store: Save/update version state
  Store-->>Service: Version result
  Service->>Queue: enqueueRagReindex()
  Service-->>Controller: Version result
  Controller-->>Client: Response
```

## Usage

```
import { Injectable } from '@nestjs/common';
import { DocVersionService } from './doc-version/doc-version.service';

@Injectable()
export class DocumentWorkflowService {
  constructor(
    private readonly docVersionService: DocVersionService,
  ) {}

  async getNextVersionPreview(): Promise<{ nextVersion: number }> {
    const nextVersion =
      await this.docVersionService.getNextVersionNumber();

    return { nextVersion };
  }

  async reindexVersionContent(): Promise<void> {
    // Use after a version state/content change when retrieval data must refresh.
    this.docVersionService.enqueueRagReindex();
  }
}
```

## AI Coding Instructions

- Keep version lifecycle logic centralized in `DocVersionService`; controllers should delegate creation, activation, scheduling, and retrieval operations to this service.
- Use `getNextVersionNumber()` when assigning new version numbers rather than calculating version values in callers.
- Trigger `enqueueRagReindex()` after changes that affect the content or active/published state available to RAG consumers.
- Preserve the distinction between active, published, and scheduled versions when modifying `activate()`, `scheduleActivation()`, or `cancelSchedule()`.
- Ensure project-scoped queries use the appropriate retrieval methods, such as `listByProject()`, rather than exposing versions across projects.

## Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `AuditService`
- DEPENDS_ON → `moduleref`
- DEPENDS_ON → `versionindexevictor`
- DEPENDS_ON → `DocVersionExportService`
- DEPENDS_ON → `ChangelogDraftsService`

## Referenced By

- `VersionCarryForwardService` (DEPENDS_ON)
- `DocVersionController` (DEPENDS_ON)
- `DocVersionModule` (MODULE_PROVIDES)
- `DocVersionModule` (MODULE_EXPORTS)
- `ScheduledPublishService` (DEPENDS_ON)
- `DocAutomationService` (DEPENDS_ON)
- `DocsTrueupService` (DEPENDS_ON)
- `TechnicalDocsMaterializerService` (DEPENDS_ON)