DocumentService

September 4, 2026

DocumentService

Kind: Service

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

`DocumentService` is the NestJS backend service responsible for managing document lifecycle operations, including creation, retrieval, updates, deletion, version history, publishing, and reverting content. It serves as the application-layer API used by controllers and other services to enforce document persistence and versioning workflows.

Methods

Method	Signature	Returns	Description
`create`	`create(dto: CreateDocumentDto, user: JwtPayload)`	`Promise<Document>`	Create a new document
`get`	`get(id: string, _user: JwtPayload)`	`Promise<Document>`	Get a single document by ID
`list`	`list(filters: DocumentFiltersDto, user: JwtPayload)`	`Promise<{ documents: Document[]; totalCount: number; limit: number; offset: number }>`	List documents with filters
`update`	`update(id: string, dto: UpdateDocumentDto, user: JwtPayload, options: { isMachineUpdate?: boolean; expectedUpdatedAt?: Date })`	`Promise<Document>`	Update a document Machine callers (parser sync, regeneration pipelines) must pass `options.isMachineUpdate` so their writes never count as manual overrides.

Method	Signature	Returns	Description
`delete`	`delete(id: string, user: JwtPayload)`	`Promise<void>`	Delete a document
`getVersions`	`getVersions(id: string, user: JwtPayload)`	`Promise<Document[]>`	Get all versions of a document
`getVersion`	`getVersion(id: string, version: number, user: JwtPayload)`	`Promise<Document>`	Get a specific version of a document
`revert`	`revert(id: string, version: number, user: JwtPayload)`	`Promise<Document>`	Revert to a previous version
`publish`	`publish(id: string, dto: PublishDocumentDto, user: JwtPayload)`	`Promise<Document>`	Publish a document
`unpublish`	`unpublish(id: string, dto: UnpublishDocumentDto, user: JwtPayload)`	`Promise<Document>`	Unpublish a document
`render`	`render(id: string, user: JwtPayload, audienceQuery: string)`	`Promise<{	`Render document with audience filtering		renderPublic	renderPublic(id: string, user: JwtPayload)	Promise<{ documentId: string; title: string; renderedContent: string; toc: TocEntry[]; viewerAudiences: string[]; createdAt: Date; createdBy: {

```
documentId: string;
title: string;
renderedContent: string;
viewerAudiences: string[];
```

name: string | null;
};
}> | Render a published document for public access (with optional authentication) |
| renderPublicByUrl | renderPublicByUrl(urlId: string, audienceFromUrl: string, user: JwtPayload) | Promise<{
documentId: string;
title: string;
slug: string;
renderedContent: string;
toc: TocEntry[];
viewerAudiences: string[];
createdAt: Date;
createdBy: {
name: string | null;
};
}> | Render a published document for public access using URL ID Supports audience filtering from URL path with security checks | | getDocumentTree | getDocumentTree(projectId: string, user: JwtPayload) | Promise<DocumentTreeNode[]> | Get document tree for authenticated users Returns a hierarchical structure of documents within a project |
getPublicDocumentTree	getPublicDocumentTree(organizationId: string, projectId: string)	Promise<DocumentTreeNode[]>	Get public document tree for unauthenticated access Returns only published public documents		bulkDelete	bulkDelete(documentIds: string[], user: JwtPayload)	Promise	Bulk delete documents
bulkMove	bulkMove(documentIds: string[], categoryId: string	null, user: JwtPayload)	Promise	Bulk move documents to a different category
bulkUpdateVisibility	bulkUpdateVisibility(documentIds: string[], isPublic: boolean, user: JwtPayload)	Promise	Bulk update document visibility
bulkPublish	bulkPublish(dto: BulkPublishDto, user: JwtPayload)	Promise<{ published: number; skipped: number; total: number }>	Bulk publish all documents in a project		duplicate	duplicate(id: string, user: JwtPayload)	Promise	Duplicate a document		compareDocuments	compareDocuments(documentId: string, fromId: string, toId: string, type: 'version' | 'revision', user: JwtPayload)	unknown	Compare two versions or revisions of a document
getDocumentStats	getDocumentStats(documentId: string, user: JwtPayload)	unknown	Get document statistics
searchDocuments	searchDocuments(filters: any, user: JwtPayload)	unknown	Enhanced search with full-text and filters
reorderDocuments	reorderDocuments(categoryId: string, documentIds: string[],

user: JwtPayload) | Promise | `Reorder documents within a category` | | reorderUncategorizedDocuments | reorderUncategorizedDocuments(documentIds: string[], user: JwtPayload) | Promise` | Reorder uncategorized documents within a project |

Dependencies

- `PrismaService`
- `UrlService`
- `TocBuilderService`
- `DocumentRevisionService`
- `DocumentVersionService`
- `DocumentIndexingService`
- `OutboxService` *(optional)*

Where it refuses work

- `DocumentService` stops the work with `NotFoundException` when `!project` — “Project not found”, in 3 places.
- `DocumentService` stops the work with `ForbiddenException` when `document.project.organizationId !== user.organizationId` — “You do not have access to this document”, in 3 places.
- `DocumentService` stops the work with `NotFoundException` when `!project` — “Project not found or access denied”, in 2 places.
- `DocumentService` stops the work with `ForbiddenException` when `!this.documentManager.canEdit(mockUser, document as any)` — “You do not have permission to edit this document”, in 2 places.
- `DocumentService` stops the work with `NotFoundException` when `!document` — “Document not found”, in 2 places.
- `DocumentService` stops the work with `BadRequestException` when `!validation.valid`.

When something fails

- `DocumentService` handles failure in 7 places: it logs it and continues in 3, discards it silently in 3, and turns it into a return value in 1. A failure discarded silently leaves no trace for whoever debugs this later.

Diagram

```
sequenceDiagram
  participant Client
  participant Controller as DocumentController
  participant Service as DocumentService
  participant Database as Document Repository

  Client->>Controller: Create or update document request
  Controller->>Service: create() / update()
  Service->>Database: Persist document and version data
  Database-->>Service: Document
  Service-->>Controller: Document response
  Controller-->>Client: JSON response

  Client->>Controller: Publish document request
  Controller->>Service: publish()
  Service->>Database: Update publication state
  Database-->>Service: Published document
  Service-->>Controller: Document response
  Controller-->>Client: JSON response
```

Usage

```
import { Injectable } from '@nestjs/common';
import { DocumentService } from './document.service';

@Injectable()
export class DocumentFacade {
  constructor(private readonly documentService: DocumentService) {}

  async publishDocument(documentId: string) {
    const document = await this.documentService.get(documentId);

    if (!document) {
      throw new Error(`Document ${documentId} was not found`);
    }

    return this.documentService.publish(documentId);
  }

  async listDocuments(limit = 20, offset = 0) {
    const result = await this.documentService.list({ limit, offset });

    return {
      documents: result.documents,
      total: result.totalCount,
      pagination: {
        limit: result.limit,
```

```
        offset: result.offset,
      },
    };
  }
}
```

AI Coding Instructions

- Use `DocumentService` as the primary integration point for document lifecycle operations; avoid accessing document persistence directly from controllers.
- Preserve versioning behavior when modifying `update()`, `revert()`, `getVersions()`, or `getVersion()` so historical document data remains consistent.
- Use `publish()` and `unpublish()` rather than directly changing publication-related fields, as these methods encapsulate publication workflow rules.
- Keep list endpoints compatible with the `{ documents, totalCount, limit, offset }` response shape for predictable pagination handling.
- Ensure authorization and ownership checks are applied at the controller or service boundary before invoking destructive operations such as `delete()` or `revert()`.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `UrlService`
- DEPENDS_ON → `TocBuilderService`
- DEPENDS_ON → `DocumentRevisionService`
- DEPENDS_ON → `DocumentVersionService`
- DEPENDS_ON → `DocumentIndexingService`
- DEPENDS_ON → `OutboxService`

Referenced By

- `DocumentController` (DEPENDS_ON)
- `CategoryDocumentsController` (DEPENDS_ON)
- `DocumentModule` (MODULE_PROVIDES)
- `DocumentModule` (MODULE_EXPORTS)
- `PublicDocumentController` (DEPENDS_ON)
- `SyncService` (DEPENDS_ON)

- `SuggestionService` (DEPENDS_ON)