# DocsVisibilityService

September 4, 2026

## DocsVisibilityService

**Kind:** Service

**Source:** `atloria-monorepo/apps/api/src/reader-access/docs-visibility.service.ts`

DocsVisibilityService — D4 rule resolution for every serving choke point.

Loads a project's visibility rules ONCE (cached 60s, like the A3 policy cache), composes the category cascade into per-document effective rules, and answers the two questions every surface asks:

- which doc ids / slugs / categories are HIDDEN from this viewer?
- which RAG visibilityKeys can this viewer's claims satisfy?

Fast path: a project with no rules costs one cached query and every filter returns null (no per-request work) — zero regression for unsegmented sites.

`DocsVisibilityService` resolves D4 documentation visibility rules for a project and viewer across every serving choke point. It loads and caches project rules for 60 seconds, applies category-to-document rule inheritance, and provides consistent filtering, authorization, and RAG visibility-key lookups without adding per-request overhead for projects with no rules.

### Methods

| Method | Signature | Returns | Description |
|---|---|---|---|
| `viewerFromRequest` | `viewerFromRequest(req: RequestWithReader, organizationId: string)` | `DocsViewer` | Build the viewer for a public request: reader-token attrs + org-member bypass. |

| Method | Signature | Returns | Description |
|---|---|---|---|
| `invalidate` | `invalidate(projectId: string)` | `void` | Drop the cached rules for a project (owner changed a rule). |
| `getRuleMap` | `getRuleMap(projectId: string)` | `Promise<ProjectRuleMap>` | |
| `projectHasRules` | `projectHasRules(projectId: string)` | `Promise<boolean>` | True when this project has any visibility rule at all (segmented site). |
| `hiddenDocIds` | `hiddenDocIds(projectId: string, viewer: DocsViewer)` | `Promise<Set | null>` | |
| `hiddenDocSlugs` | `hiddenDocSlugs(projectId: string, viewer: DocsViewer)` | `Promise<Set | null>` | |
| `hiddenCategoryIds` | `hiddenCategoryIds(projectId: string, viewer: DocsViewer)` | `Promise<Set | null>` | |
| `isDocVisible` | `isDocVisible(projectId: string, docId: string, viewer: DocsViewer)` | `Promise<boolean>` | Is one specific document visible to this viewer? |
| `assertDocVisible` | `assertDocVisible(projectId: string, docId: string, viewer: DocsViewer)` | `Promise<void>` | Consistent-404 assert for single-doc reads: a hidden doc answers with the SAME status + message as a truly missing one, so existence never leaks. |

| Method | Signature | Returns | Description |
| --- | --- | --- | --- |
| `visibilityKeysBySlug` | `visibilityKeysBySlug(projectId: string)` | `Promise<Map<string, string>>` | slug → visibilityKey for every affected published doc. |
| `satisfiableKeys` | `satisfiableKeys(projectId: string, viewer: DocsViewer)` | `Promise<string[] \| null>` | |

## Dependencies

- `PrismaService`

## Where it refuses work

- `DocsVisibilityService` stops the work with `NotFoundException` when `!(await this.isDocVisible(projectId, docId, viewer))`.
- `DocsVisibilityService` stops the work with an early return when `!map.hasRules || viewer.member`, in 5 places.
- `DocsVisibilityService` stops the work with an early return when `hit && hit.expiresAt > Date.now()`.
- `DocsVisibilityService` stops the work with an early return when `effectiveCat.has(id)`.
- `DocsVisibilityService` stops the work with an early return when `seen.has(id)`.
- `DocsVisibilityService` stops the work with an early return when `!cat`.

## Diagram

```
sequenceDiagram
  participant Request as HTTP Request
  participant Service as DocsVisibilityService
  participant Cache as Rule Cache
  participant DB as Visibility Rule Store
  participant Surface as Docs/RAG Surface

  Request->>Service: viewerFromRequest(request)
  Service-->>Request: DocsViewer (claims, identity)

  Surface->>Service: hiddenDocIds(viewer)
  Service->>Cache: getRuleMap()

  alt Cached rules available
    Cache-->>Service: ProjectRuleMap
```

```
  else Cache miss or expired (60s)
    Service->>DB: Load project + category visibility rules
    DB-->>Service: Raw rules
    Service->>Service: Compose category cascade per document
    Service->>Cache: Store ProjectRuleMap
  end

  alt Project has no visibility rules
    Service-->>Surface: null (no filtering required)
  else Project has rules
    Service->>Service: Evaluate viewer claims against effective rules
    Service-->>Surface: Hidden document/category IDs
  end

  Surface->>Service: assertDocVisible(docId, viewer)
  Service-->>Surface: Allow access or throw visibility error
```

## Usage

```
import { ForbiddenException } from '@nestjs/common';
import { DocsVisibilityService } from './docs-visibility.service';

@Injectable()
export class DocsReaderService {
  constructor(
    private readonly docsVisibility: DocsVisibilityService,
    private readonly docsRepository: DocsRepository,
  ) {}

  async getDocument(request: Request, slug: string) {
    const viewer = this.docsVisibility.viewerFromRequest(request);

    const doc = await this.docsRepository.findBySlug(slug);

    if (!doc) {
      throw new NotFoundException(`Document "${slug}" was not found`);
    }

    await this.docsVisibility.assertDocVisible(doc.id, viewer);

    return doc;
  }

  async listDocuments(request: Request) {
    const viewer = this.docsVisibility.viewerFromRequest(request);
    const hiddenIds = await this.docsVisibility.hiddenDocIds(viewer);

    const documents = await this.docsRepository.findPublished();

    // `null` means the project has no visibility rules: do not filter.
    return hiddenIds
```

```
      ? documents.filter((doc) => !hiddenIds.has(doc.id))
      : documents;
  }

  async getRagVisibilityKeys(request: Request) {
    const viewer = this.docsVisibility.viewerFromRequest(request);

    // Map<slug, visibilityKey> for documents accessible to this viewer.
    return this.docsVisibility.visibilityKeysBySlug(viewer);
  }
}
```

## AI Coding Instructions

- Always derive the viewer through `viewerFromRequest()` so all surfaces evaluate the same claims and identity inputs.
- Use `hiddenDocIds()`, `hiddenDocSlugs()`, and `hiddenCategoryIds()` for list/search filtering; treat `null` as “no project rules exist,” not as an empty hidden set.
- Use `assertDocVisible()` at direct document-serving endpoints to prevent bypassing list-level filtering through known IDs or slugs.
- Call `invalidate()` after creating, updating, or deleting project/category visibility rules so cached rule maps do not remain stale for up to 60 seconds.
- Keep RAG integrations aligned with `visibilityKeysBySlug()` rather than reimplementing claim evaluation or category cascade logic.

## Relationships

- DEPENDS_ON → `PrismaService`

## Referenced By

- `ContentVisibilityService` (DEPENDS_ON)
- `PublicProjectController` (DEPENDS_ON)
- `ReaderAccessModule` (MODULE_PROVIDES)
- `ReaderAccessModule` (MODULE_EXPORTS)
- `TechDocsRagService` (DEPENDS_ON)
- `TechnicalDocsMcpController` (DEPENDS_ON)