DocsAccessService

September 4, 2026

DocsAccessService

Kind: Service

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

DocsAccessService — the A3 access seam for reader-facing surfaces.

`assertReadable*` is the single leak-proof gate wired into DocsAccessGuard (class-level on PublicProjectController), so EVERY public surface — docs, search, llms.txt, llms-full, page.md, search-index.json, MCP, chat — stops serving for a gated project unless the caller presents a satisfying reader token or an org-member JWT. Also owns the grant flows (password/link/email) that MINT reader tokens and the per-audience visibility checks.

`DocsAccessService` is the reader-access boundary for public documentation surfaces. It resolves project policy and reader claims, enforces access through `assertReadableFromRequest`, and provides password, link, and email-code grant flows that mint reader tokens. `DocsAccessGuard` uses this service to ensure gated projects are consistently protected across docs, search, MCP, chat, LLM exports, and generated content endpoints.

Methods

Method	Signature	Returns	Description
`getPolicyContext`	`getPolicyContext(slugWithId: string)`	`Promise<PolicyContext	null>`	
`invalidate`	`invalidate()`	`void`	Drop the cache for a project (owner mutated the policy).
`resolveClaims`	`resolveClaims(req: RequestWithReader, ctx: PolicyContext)`	`Promise<ReaderClaims	null>`	
`assertReadableFromRequest`	`assertReadableFromRequest(req: RequestWithReader, res: Response)`	`Promise<void>`	The project-level gate (called by DocsAccessGuard for every :slugWithId route).
`filterVisibleAudiences`	`filterVisibleAudiences(slugWithId: string, audiences: T[], req: RequestWithReader)`	`Promise<T[]>`	Drop `hidden` audiences from a list (they must be invisible to non-members).

Method	Signature	Returns	Description
`assertAudienceReadable`	`assertAudienceReadable(slugWithId: string, audienceSlug: string, req: RequestWithReader, res: Response)`	`Promise<void>`	Guard a single audience read.
`grantPassword`	`grantPassword(slugWithId: string, password: string, ip: string, existingToken: string)`	`Promise<{ token: string }>`	POST access/password → verify bcrypt hash → mint a `password`-grant token.
`grantLink`	`grantLink(slugWithId: string, shareToken: string, existingToken: string)`	`Promise<{ token: string }>`	POST access/link → sha256 lookup, expiry/maxUses/revoked checks, useCount++ → `link:<id>` token.
`requestEmailCode`	`requestEmailCode(slugWithId: string, email: string, ip: string)`	`Promise<{ sent: true }>`	POST access/email → email a 6-digit code (allow-listed domains only).
`verifyEmailCode`	`verifyEmailCode(slugWithId: string, email: string, code: string, ip: string, existingToken: string)`	`Promise<{ token: string }>`	POST access/email/verify → check the code → mint an `email:<domain>` token.
`ssoRedirectUrl`	`ssoRedirectUrl(slugWithId: string, returnTo: string)`	`Promise<string>`	GET access/sso → the SAML reader-mode login URL to 302 to (org IdP round-trip).
`gateBranding`	`gateBranding(slugWithId: string)`	`Promise<{`	

```
projectId: string;
name: string;
mode: AccessMode;
sso: boolean;
branding: unknown;
```

}> | Ungated minimal branding for the gate screen (logo + name + colors only — NO content). |
| newShareToken | newShareToken() | { token: string; tokenHash: string }` | Random share-link token (returned once) + its stored sha256 hash. |

Dependencies

- `PrismaService`
- `ReaderTokenService`
- `RedisService`
- `EmailService` *(optional)*

Where it refuses work

- `DocsAccessService` stops the work with `NotFoundException` when `vis === 'hidden'` — “Document not found or not published for this audience”.
- `DocsAccessService` stops the work with `NotFoundException` when `!ctx` — “Project not found”.
- `DocsAccessService` stops the work with `HttpException` when `n > max` — “Too many attempts — try again shortly.”.
- `DocsAccessService` stops the work with `BadRequestException` when `ctx.mode !== 'PASSWORD'` — “This site is not password protected.”.
- `DocsAccessService` stops the work with `UnauthorizedException` when `!policy?.passwordHash || !password || !(await bcrypt.compare(password, policy.passwordHas⋯` .
- `DocsAccessService` stops the work with `BadRequestException` when `!shareToken?.trim()` — “A share token is required.”.

When something fails

- `DocsAccessService` handles failure in 1 place: it lets it reach the caller in all 1.

Diagram

```
sequenceDiagram
  participant Client as Reader Client
  participant Guard as DocsAccessGuard
  participant Access as DocsAccessService
  participant Policy as Project Policy
  participant Auth as JWT / Reader Token
  participant Surface as Public Docs Surface

  Client->>Guard: Request docs/search/MCP/chat resource
  Guard->>Access: assertReadableFromRequest()
  Access->>Policy: getPolicyContext()
  Access->>Auth: resolveClaims()

  alt Project is public
    Access-->>Guard: Access allowed
  else Valid org-member JWT or reader token
    Access->>Access: assertAudienceReadable()
    Access-->>Guard: Access allowed
  else Missing or insufficient credentials
    Access-->>Guard: Throw access denied
  end

  Guard->>Surface: Serve requested resource
  Surface-->>Client: Content response
```

Usage

```
import { Controller, Get, Req, UseGuards } from '@nestjs/common';
import { DocsAccessGuard } from './docs-access.guard';
import { DocsAccessService } from './docs-access.service';

@Controller('public/projects/:projectSlug')
@UseGuards(DocsAccessGuard)
```

```ts
export class PublicProjectController {
  constructor(private readonly docsAccessService: DocsAccessService) {}

  @Get('search-index.json')
  async getSearchIndex(@Req() request: Request) {
    // The guard already calls assertReadableFromRequest() before this handler.
    // Use audience filtering when returning content with audience restrictions.
    const pages = await this.loadProjectPages();

    return this.docsAccessService.filterVisibleAudiences(
      pages,
      (page) => page.audience,
    );
  }

  private async loadProjectPages() {
    return [
      { title: 'Getting Started', audience: 'public' },
      { title: 'Internal Runbook', audience: 'team' },
    ];
  }
}
```

AI Coding Instructions

- Route every reader-facing public surface through `DocsAccessGuard`; do not add endpoint-specific access bypasses around `assertReadableFromRequest`.
- Call `assertAudienceReadable` or `filterVisibleAudiences` when content contains per-audience visibility rules, even after project-level access succeeds.
- Use `grantPassword`, `grantLink`, and `verifyEmailCode` to mint reader tokens; avoid manually constructing or signing reader-token payloads.
- Call `invalidate()` when request-scoped policy, project configuration, or authentication state may have changed and cached access state must be refreshed.
- Preserve support for both reader tokens and organization-member JWTs when extending claim resolution or access-policy logic.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `ReaderTokenService`
- DEPENDS_ON → `RedisService`
- DEPENDS_ON → `EmailService`

Referenced By

- `PublicProjectController` (DEPENDS_ON)
- `DocsAccessGuard` (DEPENDS_ON)
- `ProjectAccessService` (DEPENDS_ON)
- `ReaderAccessModule` (MODULE_PROVIDES)

- `ReaderAccessModule` (MODULE_EXPORTS)
- `ReaderAuthGuard` (DEPENDS_ON)
- `ReaderClaimsExchangeService` (DEPENDS_ON)
- `ReaderMagicLinkService` (DEPENDS_ON)