ReaderAuthGuard

September 4, 2026

ReaderAuthGuard

Kind: Service

Source: `atloria-monorepo/apps/api/src/reader-access/reader-auth.guard.ts`

ReaderAuthGuard — the ONE shared reader-identity guard (P0-b contract).

Decodes the `X-Atloria-Reader-Token` header, verifies signature + aud + expiry + projectId(route) + ver(live tokenVersion), and attaches `request.readerClaims` (null when absent/invalid — anonymous = empty claims). It NEVER rejects: enforcement is a consumer's job (A3 = DocsAccessGuard; later C3/D4 read `readerClaims` and decide their own behaviour). This keeps the envelope reusable without changing it.

`ReaderAuthGuard` is the shared NestJS guard responsible for decoding and validating the `X-Atloria-Reader-Token` request header. It verifies token signature, audience, expiry, route project ID, and the live reader token version, then attaches validated claims to `request.readerClaims`; invalid or missing tokens become `null` rather than causing rejection. Downstream guards and handlers, such as `DocsAccessGuard`, use these claims to enforce endpoint-specific access rules.

Methods

Method	Signature	Returns
`canActivate`	`canActivate(context: ExecutionContext)`	`Promise<boolean>`

Dependencies

- `DocsAccessService`

Where it refuses work

- `ReaderAuthGuard` stops the work with an early return when `!slugWithId`.

Diagram

```
sequenceDiagram
  participant Client
  participant Guard as ReaderAuthGuard
  participant Token as Reader Token Verifier
  participant Request
  participant Consumer as DocsAccessGuard / Handler

  Client->>Guard: Request with X-Atloria-Reader-Token
  Guard->>Guard: Read route projectId and token header

  alt Token is present
    Guard->>Token: Verify signature, aud, expiry, projectId, ver
    alt Token is valid
      Token-->>Guard: Reader claims
      Guard->>Request: request.readerClaims = claims
    else Token is invalid or stale
      Token-->>Guard: Validation failure
      Guard->>Request: request.readerClaims = null
    end
  else Token is absent
    Guard->>Request: request.readerClaims = null
  end

  Guard-->>Consumer: true (never rejects)
  Consumer->>Consumer: Apply endpoint-specific authorization
```

Usage

```
import { Controller, Get, Req, UseGuards } from '@nestjs/common';
import type { Request } from 'express';
import { ReaderAuthGuard } from './reader-auth.guard';
import { DocsAccessGuard } from './docs-access.guard';

type ReaderRequest = Request & {
  readerClaims?: {
    sub: string;
    projectId: string;
    tokenVersion: number;
  } | null;
};

@Controller('projects/:projectId/docs')
@UseGuards(ReaderAuthGuard, DocsAccessGuard)
export class DocsController {
  @Get()
  async listDocs(@Req() request: ReaderRequest) {
    // ReaderAuthGuard always allows the request through.
```

```
    // DocsAccessGuard decides whether these claims grant access.
    const readerId = request.readerClaims?.sub ?? null;

    return {
      readerId,
      documents: [],
    };
  }
}
```

AI Coding Instructions

- Keep `ReaderAuthGuard` authentication-only: it must always resolve `true` and must not make authorization decisions.
- Set `request.readerClaims` to `null` for absent, malformed, expired, stale, or otherwise invalid reader tokens; do not leave stale claims on the request.
- Preserve all validation checks when changing token handling: signature, audience, expiry, route `projectId`, and live token-version (`ver`) validation.
- Apply this guard before consumers such as `DocsAccessGuard` ; downstream guards or handlers must explicitly decide how anonymous (`null`) claims are handled.
- Do not reuse reader-token validation as a substitute for user/session authentication; reader identity is a separate, optional request envelope.

Relationships

- DEPENDS_ON → `DocsAccessService`

Referenced By

- `ReaderAccessModule` (MODULE_PROVIDES)
- `ReaderAccessModule` (MODULE_EXPORTS)