ReaderClaimsExchangeService

September 4, 2026

ReaderClaimsExchangeService

Kind: Service

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

D4 claims ingestion — the third reader-identity source (docs/architecture/reader-identity.md): the customer's app signs a JWT about ITS logged-in user (role/plan/region/…); the reader posts it to `POST public/p/:slugWithId/reader/exchange`; we verify it against the project's ReaderIdentityConfig (HS256 shared secret or the customer's JWKS), map claim names via claimMap onto canonical attrs, and re-mint the ONE standard docs-reader token with `attrs` embedded (union-merged with any existing token, so grants/account survive).

Exchange-once design: the customer JWT is trusted only at this endpoint — at request time there is exactly ONE issuer (our own reader token).

`ReaderClaimsExchangeService` ingests a customer-signed JWT from the reader's application and exchanges it for Atloria's standard docs-reader token. It verifies the customer JWT using the project's `ReaderIdentityConfig`, maps configured claim names into canonical reader attributes, and union-merges those attributes with any existing reader-token attributes so grants and account context are preserved.

The customer JWT is trusted only during exchange; all subsequent requests use the single Atloria-issued reader token containing the normalized `attrs` payload.

Methods

Method	Signature	Returns
`exchange`	`exchange(slugWithId: string, customerJwt: string, existingToken: string)`	`Promise<{ token: string; attrs: Record<string, unknown> }>`

Dependencies

- `PrismaService`
- `JwtService`
- `ReaderTokenService`
- `DocsAccessService`

Where it refuses work

- `ReaderClaimsExchangeService` stops the work with `BadRequestException` when `!customerJwt || typeof customerJwt !== 'string'` — “A signed identity token is required.”.
- `ReaderClaimsExchangeService` stops the work with `NotFoundException` when `!ctx` — “Project not found”.
- `ReaderClaimsExchangeService` stops the work with `BadRequestException` when `!config?.enabled`.
- `ReaderClaimsExchangeService` stops the work with `BadRequestException` when `!config.sharedSecret` — “Reader identity is misconfigured (no shared secret).”.
- `ReaderClaimsExchangeService` stops the work with `BadRequestException` when `!config.jwksUrl` — “Reader identity is misconfigured (no JWKS URL).”.
- `ReaderClaimsExchangeService` stops the work with `Error` when `!jwk`.

When something fails

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

Diagram

```
sequenceDiagram
    participant App as Customer App
    participant API as Reader Exchange Endpoint
    participant Service as ReaderClaimsExchangeService
    participant Config as ReaderIdentityConfig
    participant JWT as JWT Verifier / Signer

    App->>API: POST /public/p/:slugWithId/reader/exchange\nCustomer-signed JWT
    API->>Service: exchange(project, customerJwt, existingReaderToken?)
    Service->>Config: Load identity provider configuration
    Service->>JWT: Verify customer JWT\nHS256 secret or customer JWKS
    JWT-->>Service: Verified customer claims
```

```
    Service->>Service: Map claims via claimMap\ninto canonical attrs
    Service->>Service: Union-merge attrs with existing token attrs
    Service->>JWT: Mint standard Atloria reader token
    JWT-->>Service: Signed reader token
    Service-->>API: { token, attrs }
    API-->>App: Standard docs-reader token
```

Usage

```
import { ReaderClaimsExchangeService } from './reader-claims-exchange.service';

@Injectable()
export class ReaderExchangeController {
  constructor(
    private readonly readerClaimsExchangeService: ReaderClaimsExchangeService,
  ) {}

  @Post('public/p/:slugWithId/reader/exchange')
  async exchangeReaderClaims(
    @Param('slugWithId') slugWithId: string,
    @Body('token') customerJwt: string,
    @Headers('authorization') authorization?: string,
  ) {
    const existingReaderToken = authorization?.replace(/^Bearer\s+/i, '');

    const result = await this.readerClaimsExchangeService.exchange(
      {
        slugWithId,
        existingReaderToken,
      },
      customerJwt,
    );

    return {
      token: result.token,
      attrs: result.attrs,
    };
  }
}
```

AI Coding Instructions

- Verify customer JWTs only through the project's `ReaderIdentityConfig`; support the configured HS256 shared secret or customer JWKS flow, never an unconfigured issuer/key.

- Apply `claimMap` before minting the internal token so downstream authorization always reads canonical `attrs` names rather than customer-specific claim names.
- Preserve existing reader-token attributes by union-merging them with exchanged attributes; do not discard grants, account identifiers, or other previously established context.
- Treat the customer JWT as exchange-only input. Downstream request authorization must trust only Atloria’s re-minted docs-reader token.
- Return both the newly minted `token` and resolved `attrs` so callers can persist the token and inspect the normalized reader identity context.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `jwtservice`
- DEPENDS_ON → `ReaderTokenService`
- DEPENDS_ON → `DocsAccessService`

Referenced By

- `PublicProjectController` (DEPENDS_ON)
- `ReaderAccessModule` (MODULE_PROVIDES)
- `ReaderAccessModule` (MODULE_EXPORTS)