ReaderAccountService

September 4, 2026

ReaderAccountService

Kind: Service

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

C3 reader accounts — the reader-facing side (whoami / usage / delete).

Identity comes EXCLUSIVELY from the verified docs-reader token (`claims.sub` = ReaderAccount.id) — never from params or body. This service can never satisfy an org-member route: it has no notion of User/organization membership at all (separate plane by construction).

`ReaderAccountService` manages reader-facing account operations for C3, including identity lookup, usage retrieval, and account deletion. It derives the reader identity exclusively from the verified docs-reader token (`claims.sub`), ensuring these endpoints cannot be used as organization-member routes or accept identity from request params or bodies.

Methods

Method	Signature	Returns	Description
`requireAccount`	`requireAccount(claims: ReaderClaims	null	undefined)`
`me`	`me(accountId: string, projectId: string)`	`Promise<{ readerId: string; email: string; signedUpAt: Date	null }>`
`usage`	`usage(accountId: string, projectId: string)`	`Promise<ReaderUsage>`	TTFC checklist + daily bars + the reader's own request log.
`deleteAccount`	`deleteAccount(accountId: string)`	`Promise<{ deleted: true }>`	Day-one right-to-erasure: deletes the GLOBAL reader

Method	Signature	Returns	Description
			account (profiles + keys cascade via FK) and ANONYMIZES the call log — rows keep their status/path/latenc…

Dependencies

- `PrismaService`

Where it refuses work

- `ReaderAccountService` stops the work with `UnauthorizedException` when `!accountId || typeof accountId !== 'string'`.
- `ReaderAccountService` stops the work with `UnauthorizedException` when `!account`.

Diagram

```
sequenceDiagram
  participant Client as Reader Client
  participant Guard as Docs Reader Auth Guard
  participant Service as ReaderAccountService
  participant DB as Database

  Client->>Guard: Request /reader-account/me|usage|delete
  Guard->>Guard: Verify docs-reader token
  Guard->>Service: Inject claims.sub as ReaderAccount.id

  Service->>Service: requireAccount()
  alt me()
    Service->>DB: Find ReaderAccount by id
    DB-->>Service: Account details
    Service-->>Client: readerId, email, signedUpAt
  else usage()
    Service->>DB: Load reader usage data
    DB-->>Service: ReaderUsage
    Service-->>Client: Usage summary
  else deleteAccount()
    Service->>DB: Delete ReaderAccount and related data
    DB-->>Service: Deletion complete
    Service-->>Client: { deleted: true }
  end
```

Usage

```
import { Controller, Delete, Get } from '@nestjs/common';
import { ReaderAccountService } from './reader-account.service';

@Controller('reader-account')
export class ReaderAccountController {
  constructor(
    private readonly readerAccountService: ReaderAccountService,
  ) {}

  @Get('me')
  async me() {
    // Identity is resolved internally from verified docs-reader claims.
    return this.readerAccountService.me();
  }

  @Get('usage')
  async usage() {
    return this.readerAccountService.usage();
  }

  @Delete()
  async deleteAccount() {
    return this.readerAccountService.deleteAccount();
  }
}
```

AI Coding Instructions

- Always resolve the active reader through verified docs-reader token claims; `claims.sub` must map directly to `ReaderAccount.id`.
- Never accept a reader/account ID from route parameters, query parameters, or request bodies for these operations.
- Use `requireAccount()` before performing reader-scoped work so unauthenticated or invalid reader contexts fail consistently.
- Keep this service isolated from `User`, organization, and organization-membership logic; it must not satisfy org-member authorization requirements.
- When adding account-related operations, preserve the reader-facing response shape and avoid exposing unrelated internal account fields.

Relationships

- DEPENDS_ON → `PrismaService`

Referenced By

- `ReaderAccountController` (DEPENDS_ON)
- `ReaderAccountModule` (MODULE_PROVIDES)
- `ReaderAccountModule` (MODULE_EXPORTS)