ReaderKeyService

September 4, 2026

ReaderKeyService

Kind: Service

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

C3 reader API keys — Atloria-issued IDENTIFIERS (`atl_rk_…`), not credentials on the customer's API. A reader key authenticates the reader at Atloria's two execution chokepoints (public playground proxy + public MCP call_operation); the upstream call is then made with the project's own server-side credential.

Storage: sha256(plaintext) only — the plaintext leaves the server exactly once, in the create response (the reader UI keeps it in docs-origin localStorage for inline samples / the playground preset).

`ReaderKeyService` manages Atloria-issued reader API key identifiers (`atl_rk_…`) used to authenticate readers at public execution entry points, including the playground proxy and MCP `call_operation` . It creates, lists, revokes, and authenticates keys while storing only a SHA-256 hash of the plaintext key; the plaintext is returned only once during creation. Upstream project credentials remain server-side and are never replaced by reader keys.

Methods

Method	Signature	Returns	Description
`create`	`create(accountId: string, projectId: string, name: string)`	`Promise<{ id: string; name: string; key: string; keyPrefix: string; createdAt: Date }>`	Create a key; the returned `key` is the ONLY time the plaintext exists outside the caller.

Method	Signature	Returns	Description
`list`	`list(accountId: string, projectId: string)`	`Promise<ReaderKeyView[]>`	The reader's own keys on this project — never the hash, never the plaintext.
`revoke`	`revoke(accountId: string, projectId: string, keyId: string)`	`Promise<{ revoked: true }>`	Revoke — scoped to the calling account + project (404 if it isn't theirs).
`authenticate`	`authenticate(rawKey: string, projectId: string)`	`Promise<AuthenticatedReaderKey | null>`	
`logPlaygroundCall`	`logPlaygroundCall(entry: { projectId: string; organizationId: string; method: string; path: string; status: number; latencyMs: number; readerId?: string | null; readerKeyId?: string | null; }) | void`	`Playground-proxy call log (source 'playground') — the same fire-and-forget + privacy discipline as mcp/exec-logging: status/method/path ONLY (path is the ups… |`	
`hash`	`hash(plaintext: string) | string`	|	

Dependencies

- `PrismaService`

Where it refuses work

- `ReaderKeyService` stops the work with `BadRequestException` when `active >= ReaderKeyService.MAX_ACTIVE_KEYS`.
- `ReaderKeyService` stops the work with `NotFoundException` when `res.count === 0` — “Key not found”.
- `ReaderKeyService` stops the work with an early return when `!rawKey || typeof rawKey !== 'string' || rawKey.length > 128 || !rawKey.startsWith(Reader…`.
- `ReaderKeyService` stops the work with an early return when `!row || row.projectId !== projectId || row.revokedAt`.

Diagram

```
sequenceDiagram
  participant Reader as Reader UI / MCP Client
  participant Service as ReaderKeyService
  participant DB as Database
  participant Proxy as Public Execution Chokepoint
  participant Upstream as Project Upstream API

  Reader->>Service: create(name)
  Service->>Service: Generate atl_rk_… plaintext key
  Service->>Service: hash(plaintext)
  Service->>DB: Store key metadata + SHA-256 hash
  Service-->>Reader: Return plaintext key once

  Reader->>Proxy: Execute request with reader key
  Proxy->>Service: authenticate(plaintext key)
  Service->>Service: hash(plaintext)
  Service->>DB: Find active key by hash
  Service-->>Proxy: AuthenticatedReaderKey | null

  alt Valid active reader key
    Proxy->>Service: logPlaygroundCall()
    Proxy->>Upstream: Call using project server-side credential
    Upstream-->>Proxy: Response
    Proxy-->>Reader: Response
  else Invalid or revoked key
    Proxy-->>Reader: Authentication error
  end
```

Usage

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { ReaderKeyService } from './reader-key.service';

@Injectable()
```

```
export class ReaderPlaygroundService {
  constructor(private readonly readerKeyService: ReaderKeyService) {}

  async createReaderKey(name: string) {
    const key = await this.readerKeyService.create(name);

    // `key.key` is plaintext and must be shown to the reader only now.
    return {
      id: key.id,
      name: key.name,
      key: key.key,
      prefix: key.keyPrefix,
      createdAt: key.createdAt,
    };
  }

  async execute(readerKey: string) {
    const authenticated = await this.readerKeyService.authenticate(readerKey);

    if (!authenticated) {
      throw new UnauthorizedException('Invalid or revoked reader key');
    }

    this.readerKeyService.logPlaygroundCall();

    // Use authenticated project context to invoke the upstream API with the
    // project's server-side credential—not the reader key.
    return {
      readerKeyId: authenticated.id,
      projectId: authenticated.projectId,
    };
  }

  async revokeReaderKey(keyId: string) {
    return this.readerKeyService.revoke(keyId);
  }
}
```

AI Coding Instructions

- Treat `atl_rk_…` values as Atloria reader identifiers, not upstream API credentials; never forward them to customer APIs.
- Return plaintext keys only from `create()` and never persist, log, or expose them again after the creation response.
- Always authenticate keys at public execution chokepoints before invoking an operation, then use the project's server-side credential for the upstream request.

- Compare and query using `hash()` output; storage must contain only the SHA-256 hash, key prefix, metadata, and revocation state.
- Ensure revoked keys fail `authenticate()` immediately, and record successful playground activity through `logPlaygroundCall()`.

Relationships

- DEPENDS_ON → `PrismaService`

Referenced By

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