OpsService

September 4, 2026

OpsService

Kind: Service

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

Admin cockpit read-side (Phase 2.3): one overview call aggregating what Tier 0/1 already
record — jobs (status/duration/failures), AI spend (AIUsage), credits, queue health and
the F2 interaction stream (deflection). Org-scoped: you see your own organization.

`OpsService` powers the organization-scoped admin cockpit overview for Phase 2.3. It aggregates operational read-side data—including job health, AI usage and spend, credit balances, queue status, and F2 interaction/deflection activity—into a single dashboard-oriented response. The service is intended for observability and operational decision-making rather than mutating underlying records.

Methods

Method	Signature	Returns
`overview`	`overview(organizationId: string)`	`unknown`

Dependencies

- `PrismaService`
- `ConfigService`

When something fails

- `OpsService` handles failure in 1 place: it logs it and continues in all 1.

Diagram

```
sequenceDiagram
  participant Admin as Admin Client
  participant Controller as Ops Controller
  participant Ops as OpsService
  participant Jobs as Jobs Data
  participant AI as AIUsage Data
  participant Credits as Credits Data
  participant Queue as Queue Health
  participant F2 as F2 Interaction Stream

  Admin->>Controller: Request organization overview
  Controller->>Ops: getOverview(orgId)
  Ops->>Jobs: Fetch job status, duration, failures
  Ops->>AI: Fetch usage and spend totals
  Ops->>Credits: Fetch credit information
  Ops->>Queue: Fetch queue health metrics
  Ops->>F2: Fetch interaction and deflection metrics

  Jobs-->>Ops: Job metrics
  AI-->>Ops: AI spend metrics
  Credits-->>Ops: Credit metrics
  Queue-->>Ops: Queue metrics
  F2-->>Ops: Deflection metrics

  Ops-->>Controller: Aggregated ops overview
  Controller-->>Admin: Organization-scoped dashboard data
```

Usage

```
import { Controller, Get, Req } from '@nestjs/common';
import { OpsService } from './ops.service';

@Controller('ops')
export class OpsController {
  constructor(private readonly opsService: OpsService) {}

  @Get('overview')
  async getOverview(@Req() request: { user: { organizationId: string } }) {
    // Always derive the organization scope from the authenticated user.
    return this.opsService.getOverview(request.user.organizationId);
  }
}
```

AI Coding Instructions

- Keep all queries organization-scoped; never accept an arbitrary organization ID without authorization checks.

- Treat this as a read-side aggregation service: avoid creating or updating jobs, usage records, credits, or interaction events here.
- Preserve a stable overview response shape, since admin cockpit consumers may depend on metric names and defaults.
- Fetch independent metric groups concurrently where possible, while ensuring partial or unavailable operational data is handled safely.
- Reuse the existing job, AI usage, credit, queue, and F2 data-access services rather than duplicating their query logic.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `configservice`

Referenced By

- `OpsController` (DEPENDS_ON)
- `OpsModule` (MODULE_PROVIDES)