JobReconcileService

September 4, 2026

JobReconcileService

Kind: Service

Source: `atloria-monorepo/apps/api/src/documentation/services/job-reconcile.service.ts`

Job Reconciliation — the honesty guarantee for documentation generation jobs.

A DocumentationJob goes PENDING → RUNNING and the external atlorix-runner POSTs
progress back to /jobs/:id/external-progress, bumping `updatedAt` on every report.
If that subprocess dies WITHOUT calling /jobs/:id/external-failed, the row is orphaned at RUNNING with whatever progress it last reported — the infamous "Generation in progress (5%)" that never moves and never fails.

Before this service the ONLY reaper was inline in DocAutomationService.startJob(): it auto-expired stale RUNNING jobs, but only when a NEW job was started for the SAME project. So if nobody kicked off another run, an orphan could sit at 5% forever (verified live: a week-old 5% job on a real project's Overview).

This closes the gap two ways, so status can never lie:

1. reconcile(scope) — the shared core. Any PENDING/RUNNING job whose `updatedAt`
   (its progress heartbeat) is older than the stall threshold is transitioned to FAILED with a clear, honest reason. Healthy jobs (recent heartbeat) and jobs already terminal (COMPLETED/FAILED/CANCELLED) are never touched. Called defensively at READ time by listJobs()/getJob() so a stale row can never even be *served* as in-progress — best-effort, it never throws into the read path.
2. sweep() — an

`JobReconcileService` prevents documentation jobs from remaining falsely marked as `PENDING` or `RUNNING` after their external runner has stopped reporting progress. It detects jobs whose `updatedAt` heartbeat exceeds the configured stall threshold,

transitions them to `FAILED` with an honest failure reason, and supports both defensive read-time reconciliation and periodic background sweeps.

Methods

Method	Signature	Returns	Description
`stallMinutes`	`stallMinutes()`	`number`	Stall threshold in minutes, tunable via DOC_JOB_STALL_MINUTES (floored at 1).
`reconcile`	`reconcile(scope: { organizationId?: string; projectId?: string; id?: string })`	`Promise<number>`	Reconcile stalled jobs → FAILED.
`sweep`	`sweep()`	`Promise<void>`	Periodic global sweep — one replica per tick, so nothing sits stale unseen.
`onModuleDestroy`	`onModuleDestroy()`	`void`	

Dependencies

- `PrismaService`
- `ConfigService`

Where it refuses work

- `JobReconcileService` stops the work with an early return when `!Number.isFinite(raw) || raw <= 0`.
- `JobReconcileService` stops the work with an early return when `process.env['DOC_JOB_RECONCILE_DISABLED'] === 'true'`.
- `JobReconcileService` stops the work with an early return when `!(await this.acquireLock())`.

When something fails

- `JobReconcileService` handles failure in 2 places: it turns it into a return value in all 2.

Diagram

```
sequenceDiagram
  participant Client
  participant JobsAPI as Jobs API
  participant Reconciler as JobReconcileService
  participant DB as Database
  participant Runner as atlorix-runner

  Runner->>JobsAPI: POST /jobs/:id/external-progress
  JobsAPI->>DB: Update progress and updatedAt

  Client->>JobsAPI: GET /jobs or /jobs/:id
  JobsAPI->>Reconciler: reconcile()
  Reconciler->>DB: Find stale PENDING/RUNNING jobs
  Reconciler->>DB: Mark stale jobs as FAILED
  JobsAPI->>DB: Read current job status
  JobsAPI-->>Client: Return reconciled status

  loop Periodic sweep
    Reconciler->>Reconciler: sweep()
    Reconciler->>DB: Reconcile stale jobs
  end
```

Usage

```
import { Injectable } from '@nestjs/common';
import { JobReconcileService } from './job-reconcile.service';

@Injectable()
export class DocumentationJobsService {
  constructor(
    private readonly jobReconcileService: JobReconcileService,
  ) {}

  async listJobs() {
    // Best-effort reconciliation before serving job statuses.
    await this.jobReconcileService.reconcile();

    // Fetch and return jobs after stale in-progress records are failed.
    return this.fetchJobsFromDatabase();
  }

  async runMaintenanceSweep(): Promise<void> {
    // Intended for scheduled/background maintenance.
    await this.jobReconcileService.sweep();
  }

  private async fetchJobsFromDatabase() {
    return [];
```

```
  }
}
```

AI Coding Instructions

- Treat `updatedAt` as the job heartbeat: external progress reports must update it so healthy long-running jobs are not incorrectly failed.
- Only reconcile non-terminal jobs (`PENDING` and `RUNNING`); never modify `COMPLETED` , `FAILED` , or `CANCELLED` jobs.
- Keep reconciliation best-effort when called from read paths—failures in `reconcile()` must not prevent `listJobs()` or `getJob()` from returning data.
- Use `sweep()` for scheduled/background cleanup and ensure timers or scheduler resources are stopped through `onModuleDestroy()` .
- When adding job states or runner callbacks, update reconciliation rules so stale jobs cannot be served indefinitely as in progress.

Relationships

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

Referenced By

- `DocumentationModule` (MODULE_PROVIDES)
- `DocAutomationService` (DEPENDS_ON)