DocsPrQueue

September 4, 2026

DocsPrQueue

Kind: Service

Source: `atloria-monorepo/apps/api/src/docs-pr/docs-pr-queue.service.ts`

Durable queue for the automatic docs-PR flow. Turns the webhook's fire-and-forget run into a
Redis-backed job: it survives pod restarts, retries on transient failures (a GitHub blip), and
is observable. The worker runs in-process (low volume, seconds per job). Redundant jobs are
harmless — the adapter's deterministic branch makes every run idempotent.

The processor is registered by DocsPrService (callback, not DI) so there's no module cycle.
If Redis is unavailable, `enqueue` returns false and the caller runs inline (behavior preserved).

`DocsPrQueue` provides a Redis-backed, durable job queue for the automatic documentation pull request workflow. It accepts webhook-triggered work, retries transient failures, and runs a registered in-process processor while preserving the existing inline fallback when Redis is unavailable. The processor is registered by `DocsPrService` via callback to avoid a NestJS module dependency cycle.

Methods

Method	Signature	Returns	Description
`registerProcessor`	`registerProcessor(fn: (job: DocsPrJob) => Promise<void>)`	`void`	
`enqueue`	`enqueue(job: DocsPrJob)`	`Promise<boolean>`	Durable enqueue; returns false if the queue is unavailable

Method	Signature	Returns	Description
			so the caller can run inline.
`onModuleInit`	`onModuleInit()`	`unknown`	
`onModuleDestroy`	`onModuleDestroy()`	`unknown`	

Dependencies

- `ConfigService`

Where it refuses work

- `DocsPrQueue` stops the work with an early return when `!this.queue`.

When something fails

- `DocsPrQueue` handles failure in 2 places: it logs it and continues in 1, and turns it into a return value in 1.

Diagram

```
sequenceDiagram
  participant Webhook as GitHub Webhook
  participant Service as DocsPrService
  participant Queue as DocsPrQueue
  participant Redis as Redis Queue
  participant Worker as In-process Worker
  participant GitHub as GitHub API

  Webhook->>Service: docs PR event
  Service->>Queue: enqueue(job)

  alt Redis available
    Queue->>Redis: persist job
    Redis-->>Queue: job accepted
    Queue-->>Service: true
    Worker->>Redis: claim next job
    Worker->>Queue: run registered processor
    Queue->>GitHub: create/update deterministic PR branch
    GitHub-->>Queue: result
  else Redis unavailable
    Queue-->>Service: false
    Service->>GitHub: run docs PR flow inline
  end
```

Usage

```
import { Injectable, OnModuleInit } from '@nestjs/common';
import { DocsPrQueue } from './docs-pr-queue.service';

@Injectable()
export class DocsPrService implements OnModuleInit {
  constructor(private readonly docsPrQueue: DocsPrQueue) {}

  onModuleInit(): void {
    this.docsPrQueue.registerProcessor(async () => {
      // Run the idempotent docs PR workflow here.
      // The GitHub adapter should use a deterministic branch name so retries
      // and duplicate jobs safely update the same pull request.
      await this.createOrUpdateDocsPullRequest();
    });
  }

  async handleWebhook(): Promise<void> {
    const queued = await this.docsPrQueue.enqueue();

    if (!queued) {
      // Preserve webhook behavior when Redis or the queue is unavailable.
      await this.createOrUpdateDocsPullRequest();
    }
  }

  private async createOrUpdateDocsPullRequest(): Promise<void> {
    // Generate docs, push the deterministic branch, and create/update the PR.
  }
}
```

AI Coding Instructions

- Register the processor from `DocsPrService` using `registerProcessor()` rather than injecting `DocsPrService` into the queue; this prevents a NestJS module cycle.
- Treat `enqueue()` returning `false` as an expected Redis-unavailable path and execute the docs PR flow inline to preserve webhook behavior.
- Keep processor work idempotent: duplicate queue jobs and retries must safely update the same deterministic GitHub branch/PR.
- Do not move long-running work into the webhook handler when queueing succeeds; let the in-process worker claim and process persisted jobs.
- Ensure lifecycle hooks remain wired so queue resources and worker processing are initialized and shut down through NestJS module lifecycle events.

Relationships

- DEPENDS_ON → `configservice`

Referenced By

- `DocsPrModule` (MODULE_PROVIDES)
- `DocsPrService` (DEPENDS_ON)