ChangeRequestService

September 4, 2026

ChangeRequestService

Kind: Service

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

ChangeRequestService — DB lifecycle for docs change requests (S2.1).

A change request = a git branch (cr/) holding proposed content + a DocsChangeRequest row
for metadata/status. This service ONLY touches the DB and the queue: api pods cannot reach
the git repo (RWO PVC on the worker), so `open` persists a 'draft' row carrying the proposed
content and enqueues a 'cr-open' job; CrWorker stages the branch and flips the row to 'open'
(markOpen) or 'failed' (markFailed).

`ChangeRequestService` manages the database lifecycle for documentation change requests and coordinates background work through the queue. API pods create draft records and enqueue work, while `CrWorker` performs Git operations on the worker-mounted repository and updates requests to `open` or `failed`.

Methods

Method	Signature	Returns	Description
`open`	`open(projectId: string, dto: OpenChangeRequestDto, author: { authorType: string; authorId?: string | null })`	`unknown`	
`list`	`list(projectId: string, status: string, take: unknown)`	`unknown`	List a project's change requests, newest first, optionally filtered by status.

Method	Signature	Returns	Description
`get`	`get(projectId: string, crId: string)`	`unknown`	One CR, scoped to the project (a crId from another project is a 404, not a leak).
`merge`	`merge(projectId: string, crId: string, mergedById: string)`	`unknown`	Trigger the merge: ATOMIC open → 'merging' (the guard lives in the WHERE clause — a concurrent close/merge can never be clobbered), then enqueue the 'cr-merg…
`refreshDiff`	`refreshDiff(projectId: string, crId: string)`	`unknown`	Recompute diffCache + conflictState against CURRENT main without merging anything — status stays untouched; the UI polls the row for the refreshed result.
`resolve`	`resolve(projectId: string, crId: string, dto: ResolveChangeRequestDto, resolvedById: string	`null)`	`unknown`
`close`	`close(projectId: string, crId: string)`	`unknown`	Close without merging.
`markOpen`	`markOpen(crId: string, data: {		

```
  baseCommit: string | null;
  headCommit: string;
  files?: CrCommittedFile[];
  diffCache?: CrDiff | null;
  previewFiles?: CrPreviewFile[];
})` | `unknown` | Worker hook: the branch is staged — record the commits, swap `files` to its c
```

| `markFailed` | `markFailed(crId: string, error: string)` | `unknown` | Worker hook: record a failure. |

| `reclaimMerging` | `reclaimMerging(crId: string)` | `Promise<boolean>` | Worker hook: a BullMQ retry re-driving a merge that failed mid-writeback reclaims the row 'failed' → 'merging'. |

| `markMerged` | `markMerged(crId: string)` | `Promise<boolean>` | Worker hook: writeback completed — 'merging' → 'merged' with mergedAt. |

| `abortMergeWithConflicts` | `abortMergeWithConflicts(crId: string, conflictState: CrConflictState, error: string)` | `unknown` | Worker hook: the conflict gate (or a mid-

writeback CAS loss) found the merge unsafe — back to 'open' with the conflict report persisted. |
`markMergeFailed`	`markMergeFailed(crId: string, error: string)`	`unknown`	Worker hook: merge failed mid-writeback — 'merging' → 'failed' (atomic; swallow-and-log like markFailed since this runs on error paths).
`recordRefresh`	`recordRefresh(crId: string, diffCache: CrDiff, conflictState: CrConflictState | null)`	`unknown`	Worker hook: a cr-refresh recompute landed — store it; status untouched.
`recordRestage`	`recordRestage(crId: string, data: { baseCommit: string | null; headCommit: string; diffCache: CrDiff; conflictState: CrConflictState | null; previewFiles?: CrPreviewFile[]; })`	`unknown`	Worker hook: a cr-restage committed the resolution — the branch advanced (new base/head), the diff/conflict caches are fresh, and the pendingResolution is co…

Dependencies

- `PrismaService`
- `DocsCrQueue`

Where it refuses work

- `ChangeRequestService` stops the work with `BadRequestException` when `row.status !== 'open'`, in 2 places.
- `ChangeRequestService` stops the work with `BadRequestException` when `totalBytes >= MAX_PROPOSED_BYTES`.
- `ChangeRequestService` stops the work with `BadRequestException` when `file.createPath || file.delete` — “createPath/delete file entries are only valid on git-sync change requests”.
- `ChangeRequestService` stops the work with `BadRequestException` when `!file.documentId` — “each file needs a documentId”.
- `ChangeRequestService` stops the work with `BadRequestException` when `!file.documentId` — “a delete entry needs the documentId to unpublish”.
- `ChangeRequestService` stops the work with `BadRequestException` when `!file.documentId && !file.createPath` — “each git file entry needs a documentId or a createPath”.

When something fails

- `ChangeRequestService` handles failure in 3 places: it turns it into a return value in 2, and discards it silently in 1. A failure discarded silently leaves no trace for whoever debugs this later.

Diagram

```
sequenceDiagram
  participant Client
  participant API as ChangeRequestService
  participant DB as DocsChangeRequest DB
  participant Queue
  participant Worker as CrWorker
  participant Git as Git Repository

  Client->>API: open(proposed content)
  API->>DB: Create request with status "draft"
  API->>Queue: Enqueue "cr-open" job
  API-->>Client: Return draft request

  Queue->>Worker: Process cr-open job
  Worker->>Git: Create branch cr/<id> and stage content
  alt Branch staged successfully
    Worker->>API: markOpen(id)
    API->>DB: Update status to "open"
  else Git operation failed
    Worker->>API: markFailed(id, error)
    API->>DB: Update status to "failed"
  end

  Client->>API: merge(), resolve(), or close()
  API->>DB: Update request lifecycle state
```

Usage

```
import { ChangeRequestService } from './change-request.service';

@Injectable()
export class DocsController {
  constructor(
    private readonly changeRequests: ChangeRequestService,
  ) {}

  async createChangeRequest(userId: string) {
    const request = await this.changeRequests.open({
      authorId: userId,
      documentId: 'getting-started',
      title: 'Clarify installation instructions',
```

```
      content: '# Getting Started\n\nUpdated installation content.',
    });

    // The request is initially a draft while CrWorker creates cr/<id>.
    return request;
  }

  async getOpenRequests() {
    return this.changeRequests.list({
      status: 'open',
    });
  }

  async closeRequest(requestId: string, userId: string) {
    return this.changeRequests.close(requestId, userId);
  }
}
```

AI Coding Instructions

- Keep Git repository operations out of this service; API pods only persist request state and enqueue jobs.
- Treat `draft` as the pre-worker state: `open()` must create the DB record before the `cr-open` job is processed.
- Use `markOpen()` and `markFailed()` from worker flows after branch staging succeeds or fails; preserve failure details when available.
- Respect lifecycle transitions when implementing `merge()`, `resolve()`, and `close()` so requests cannot be acted on from invalid states.
- Use `reclaimMerging()` for recovery of requests left in a merging state after interrupted worker or deployment activity.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `DocsCrQueue`

Referenced By

- `ChangeRequestController` (DEPENDS_ON)
- `ChangeRequestModule` (MODULE_PROVIDES)
- `ChangeRequestModule` (MODULE_EXPORTS)
- `DocAutomationService` (DEPENDS_ON)

- `GitSyncIngestService` (DEPENDS_ON)