OutboxService

September 4, 2026

OutboxService

Kind: Service

Source: `atloria-monorepo/apps/api/src/docs-repo/outbox.service.ts`

OutboxService — the transactional outbox for the docs substrate.

`record` is called AFTER a successful content mutation (publish/reconcile/regen). It is fire-and-forget and RESILIENT: it writes ONE row and NEVER throws into the caller — a
failed outbox insert must not fail the request that already succeeded. The committer worker drains these rows and coalesces them into repo commits.

`OutboxService` implements the transactional outbox for the documentation substrate. After a successful content mutation—such as publish, reconcile, or regeneration—it records a durable event without allowing outbox failures to affect the completed request. A committer worker later drains pending events, coalesces them into repository commits, and marks or retries processed rows.

Methods

Method	Signature	Returns	Description
`record`	`record(projectId: string, eventType: string, payload: Record<string, unknown>)`	`Promise<void>`	Record a single outbox event.
`drainPending`	`drainPending(limit: unknown)`	`Promise<OutboxBatch[]>`	Read up to `limit` unprocessed rows (oldest first) grouped by project — the unit the

Method	Signature	Returns	Description
			committer coalesces into one commit per project.
`pendingForProject`	`pendingForProject(projectId: string, limit: unknown)`	`Promise<OutboxEvent[]>`	All unprocessed rows for one project (oldest first) — the committer's drain unit.
`pendingDepth`	`pendingDepth()`	`Promise<number>`	Total unprocessed rows across all projects — the outbox-depth health signal (backlog size).
`markProcessed`	`markProcessed(ids: string[])`	`Promise<void>`	Mark rows processed once their coalesced commit landed.
`incrementAttempts`	`incrementAttempts(ids: string[])`	`Promise<void>`	Bump the retry counter for rows whose coalesced commit FAILED.

Dependencies

- `PrismaService`

Where it refuses work

- `OutboxService` stops the work with an early return when `!ids.length`, in 2 places.

When something fails

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

Diagram

```
sequenceDiagram
  participant Mutation as Content Mutation
  participant Outbox as OutboxService
  participant DB as Outbox Storage
  participant Worker as Committer Worker
  participant Repo as Docs Repository

  Mutation->>Mutation: Publish / reconcile / regenerate succeeds
  Mutation->>Outbox: record(event)
  Outbox->>DB: Insert one pending outbox row
  alt Insert fails
    Outbox-->>Mutation: Resolve without throwing
  else Insert succeeds
    Outbox-->>Mutation: Resolve
  end

  Worker->>Outbox: drainPending()
  Outbox->>DB: Fetch pending events grouped into batches
  DB-->>Outbox: OutboxBatch[]
  Outbox-->>Worker: Pending batches

  Worker->>Repo: Coalesce and commit batch changes
  alt Commit succeeds
    Worker->>Outbox: markProcessed(eventIds)
    Outbox->>DB: Mark rows processed
  else Commit fails
    Worker->>Outbox: incrementAttempts(eventIds)
    Outbox->>DB: Increment retry attempts
  end
```

Usage

```
import { Injectable } from '@nestjs/common';
import { OutboxService } from './outbox.service';

@Injectable()
export class DocsPublishService {
```

```
  constructor(private readonly outboxService: OutboxService) {}

  async publishDocument(projectId: string, documentId: string): Promise<void> {
    // Perform and complete the primary content mutation first.
    await this.publishContent(projectId, documentId);

    // Fire-and-forget resilient recording: this must not fail publishing.
    void this.outboxService.record({
      projectId,
      documentId,
      type: 'document.published',
    });
  }

  private async publishContent(
    projectId: string,
    documentId: string,
  ): Promise<void> {
    // Persist published document content.
  }
}

// Example committer worker flow:
async function processOutbox(outboxService: OutboxService) {
  const batches = await outboxService.drainPending();

  for (const batch of batches) {
    try {
      await commitBatchToRepository(batch);
      await outboxService.markProcessed(batch.events.map((event) => event.id));
    } catch {
      await outboxService.incrementAttempts(
        batch.events.map((event) => event.id),
      );
    }
  }
}
```

AI Coding Instructions

- Call `record()` only after the primary publish, reconcile, or regeneration mutation has completed successfully.
- Treat `record()` as resilient fire-and-forget behavior; do not make a successful content request depend on an outbox insert succeeding.
- Use `drainPending()` from the committer worker, then coalesce each returned `OutboxBatch` into as few repository commits as possible.

- Call `markProcessed()` only after the corresponding repository commit succeeds; call `incrementAttempts()` when processing or committing fails.
- Use `pendingForProject()` and `pendingDepth()` for diagnostics, monitoring, and backpressure visibility rather than as the primary worker-drain mechanism.

Relationships

- DEPENDS_ON → `PrismaService`

Referenced By

- `CollaborationService` (DEPENDS_ON)
- `DocsRepoModule` (MODULE_PROVIDES)
- `DocsRepoModule` (MODULE_EXPORTS)
- `OutboxPollerService` (DEPENDS_ON)
- `DocumentService` (DEPENDS_ON)
- `DocAutomationService` (DEPENDS_ON)
- `TranslationsService` (DEPENDS_ON)
- `SnippetsService` (DEPENDS_ON)