ChangelogSubscriptionsService

September 4, 2026

ChangelogSubscriptionsService

Kind: Service

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

Double-opt-in changelog subscriptions + the publish email fan-out (B5).

pending → (emailed confirm token clicked) → confirmed → (unsubscribe token) → unsubscribed. The tokens ARE the capability: confirm/unsubscribe links must work straight from an email client with no reader auth, so the public controller mounts those two routes with

`ChangelogSubscriptionsService` manages double-opt-in subscriptions for project changelog emails and sends published entries to confirmed subscribers. It owns the subscription lifecycle— `pending` → `confirmed` → `unsubscribed` —using capability tokens embedded in email links, so confirmation and unsubscribe actions do not require authenticated users.

Methods

Method	Signature	Returns	Description
`subscribe`	`subscribe(projectId: string, rawEmail: string)`	`Promise<{ status: 'pending' }>`	Start (or restart) a double-opt-in subscription.
`confirm`	`confirm(token: string)`	`Promise<{ projectName: string }>`	Redeem a confirm token (from the emailed link).
`unsubscribe`	`unsubscribe(token: string)`	`Promise<{ projectName: string }>`	Redeem an unsubscribe token (from the footer of every fan-out email).
`fanoutEntry`	`fanoutEntry(entryId: string)`	`Promise<void>`	Email every confirmed subscriber about a freshly published entry.

Dependencies

- `PrismaService`
- `EmailService`

Where it refuses work

- `ChangelogSubscriptionsService` stops the work with `NotFoundException` when `!project` — “Project not found”.
- `ChangelogSubscriptionsService` stops the work with `BadRequestException` when `!EMAIL_RE.test(email) || email.length > 320` — “Invalid email address”.
- `ChangelogSubscriptionsService` stops the work with `NotFoundException` when `!sub` — “Invalid or expired confirmation link”.
- `ChangelogSubscriptionsService` stops the work with `NotFoundException` when `!sub` — “Invalid unsubscribe link”.
- `ChangelogSubscriptionsService` stops the work with an early return when `existing?.status === 'confirmed'`.
- `ChangelogSubscriptionsService` stops the work with an early return when `!entry || entry.status !== 'published'`.

When something fails

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

Diagram

```
sequenceDiagram
  participant Visitor
  participant Service as ChangelogSubscriptionsService
  participant DB as Database
  participant Email as Email Provider
  participant Subscriber

  Visitor->>Service: subscribe(projectId, email)
  Service->>DB: Create/update pending subscription with confirm token
  Service->>Email: Send confirmation email with confirm link
  Service-->>Visitor: { status: "pending" }

  Subscriber->>Service: confirm(confirmToken)
  Service->>DB: Validate token and mark subscription confirmed
  Service-->>Subscriber: { projectName }
```

```
  Service->>Service: fanoutEntry(entry)
  Service->>DB: Load confirmed subscriptions for project
  loop Each confirmed subscriber
    Service->>Email: Send changelog entry with unsubscribe token
  end

  Subscriber->>Service: unsubscribe(unsubscribeToken)
  Service->>DB: Validate token and mark subscription unsubscribed
  Service-->>Subscriber: { projectName }
```

Usage

```
import { ChangelogSubscriptionsService } from './changelog-subscriptions.service';

@Injectable()
export class ChangelogController {
  constructor(
    private readonly subscriptions: ChangelogSubscriptionsService,
  ) {}

  @Post(':projectId/subscribe')
  async subscribe(
    @Param('projectId') projectId: string,
    @Body('email') email: string,
  ) {
    return this.subscriptions.subscribe(projectId, email);
    // Returns: { status: 'pending' }
  }

  @Get('subscriptions/confirm/:token')
  async confirm(@Param('token') token: string) {
    return this.subscriptions.confirm(token);
    // Returns: { projectName: 'My Project' }
  }

  @Get('subscriptions/unsubscribe/:token')
  async unsubscribe(@Param('token') token: string) {
    return this.subscriptions.unsubscribe(token);
    // Returns: { projectName: 'My Project' }
  }
}

// Invoke after publishing a changelog entry.
await changelogSubscriptionsService.fanoutEntry(publishedEntry);
```

AI Coding Instructions

- Treat confirmation and unsubscribe tokens as capabilities: the public routes must validate tokens without requiring reader authentication.
- Preserve the subscription state machine; only confirmed subscriptions should receive changelog fan-out emails.
- Generate and store distinct, securely random confirmation and unsubscribe tokens; do not expose subscription identifiers in email URLs.
- Call `fanoutEntry()` only after an entry is successfully published, and ensure failures for individual recipients do not prevent delivery to others.
- Keep confirmation and unsubscribe operations idempotent so email-link retries, scanners, and repeated clicks are handled safely.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `EmailService`

Referenced By

- `ChangelogDraftsService` (DEPENDS_ON)
- `ChangelogModule` (MODULE_PROVIDES)
- `PublicChangelogController` (DEPENDS_ON)