NotificationService

September 4, 2026

NotificationService

Kind: Service

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

`NotificationService` manages creation, delivery, retrieval, and read-state updates for user notifications in the API. It centralizes notification flows for mentions, new comments, and suggestions, and provides user-facing queries such as unread notifications and notification lists.

Methods

Method	Signature	Returns	Description
`createNotification`	`createNotification(dto: CreateNotificationDto)`	`unknown`	Create a notification for a user
`notifyMentions`	`notifyMentions(comment: { id: string; content: string; documentId: string }, mentionedUserIds: string[], authorName: string)`	`unknown`	Notify users who were mentioned in a comment
`notifyNewComment`	`notifyNewComment(documentId: string, commentId: string, authorName: string, subscribedUserIds: string[])`	`unknown`	Notify when a new comment is added to a document
`notifySuggestion`	`notifySuggestion(documentId: string, suggestionId: string, authorName: string, documentOwnerId: string)`	`unknown`	Notify when a suggestion is made
`getUnreadNotifications`	`getUnreadNotifications(userId: string)`	`unknown`	Get unread notifications for a user

Method	Signature	Returns	Description
`listForUser`	`listForUser(userId: string, opts: ListNotificationsOptions)`	`unknown`	List a user's notification feed for the navbar bell (P1.d reader API): newest first, paginated, optional unread-only filter.
`markAsRead`	`markAsRead(notificationId: string, userId: string)`	`unknown`	Mark notification as read.
`markAllAsRead`	`markAllAsRead(userId: string)`	`unknown`	Mark all notifications as read for a user
`getUserNotifications`	`getUserNotifications(userId: string, limit: unknown)`	`unknown`	Get all notifications for a user

Dependencies

- `PrismaService`

Where it refuses work

- `NotificationService` stops the work with `NotFoundException` when `result.count === 0` — “Notification not found”.

When something fails

- `NotificationService` handles failure in 1 place: it lets it reach the caller in all 1.

Diagram

```
sequenceDiagram
  participant Client
  participant Feature as Comment/Suggestion Feature
  participant Notifications as NotificationService
  participant Store as Notification Repository
```

```
  Feature->>Notifications: notifyMentions() / notifyNewComment() / notifySuggestion()
  Notifications->>Notifications: createNotification()
  Notifications->>Store: Persist notification
  Store-->>Notifications: Created notification

  Client->>Notifications: getUnreadNotifications() / listForUser()
  Notifications->>Store: Query user notifications
  Store-->>Notifications: Notifications and unread count
  Notifications-->>Client: Notification results

  Client->>Notifications: markAsRead() / markAllAsRead()
  Notifications->>Store: Update read state
  Store-->>Notifications: Updated notification(s)
```

Usage

```
import { Controller, Get, Param, Patch } from '@nestjs/common';
import { NotificationService } from './notification.service';

@Controller('notifications')
export class NotificationController {
  constructor(
    private readonly notificationService: NotificationService,
  ) {}

  @Get('unread')
  async getUnread() {
    return this.notificationService.getUnreadNotifications();
  }

  @Get()
  async listForCurrentUser() {
    return this.notificationService.getUserNotifications();
  }

  @Patch(':notificationId/read')
  async markRead(@Param('notificationId') notificationId: string) {
    return this.notificationService.markAsRead(notificationId);
  }

  @Patch('read-all')
  async markAllRead() {
    return this.notificationService.markAllAsRead();
  }
}
```

AI Coding Instructions

- Use `createNotification()` as the shared creation path; feature-specific methods such as `notifyNewComment()` should delegate to it rather than duplicating persistence logic.
- Trigger notification methods from the relevant domain workflow only after the related comment, mention, or suggestion has been successfully created.
- Ensure notification queries and read-state updates are scoped to the authenticated user to prevent cross-user notification access.
- Prefer `markAllAsRead()` for bulk read-state updates instead of iterating through notifications in application code.
- Keep notification payloads consistent across mention, comment, and suggestion flows so clients can render them predictably.

Relationships

- DEPENDS_ON → `PrismaService`

Referenced By

- `CommentService` (DEPENDS_ON)
- `DocAutomationService` (DEPENDS_ON)
- `NotificationController` (DEPENDS_ON)
- `NotificationModule` (MODULE_PROVIDES)
- `NotificationModule` (MODULE_EXPORTS)
- `SuggestionService` (DEPENDS_ON)
- `TechnicalDocsGenerationService` (DEPENDS_ON)