TemplateService

September 4, 2026

TemplateService

Kind: Service

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

Service for managing document templates.
Provides CRUD operations and template usage tracking.

`TemplateService` manages reusable document templates in the API layer. It provides CRUD operations, creates documents from templates, saves existing documents as templates, and supports template usage tracking within the document workflow.

Methods

Method	Signature	Returns	Description
`list`	`list(filters: TemplateFiltersDto, user: JwtPayload)`	`Promise<DocumentTemplate[]>`	List all templates with optional filters.
`getById`	`getById(id: string, user: JwtPayload)`	`Promise<DocumentTemplate>`	Get a single template by ID.
`create`	`create(dto: CreateTemplateDto, user: JwtPayload)`	`Promise<DocumentTemplate>`	Create a new template.
`createDocumentFromTemplate`	`createDocumentFromTemplate(templateId: string, dto: CreateFromTemplateDto, user: JwtPayload)`	`Promise<Document>`	Create a new document from a template.
`saveAsTemplate`	`saveAsTemplate(documentId: string, dto: SaveAsTemplateDto, userId: string)`	`Promise<DocumentTemplate>`	Save an existing document as a new template.
`update`	`update(id: string, dto: Partial<CreateTemplateDto>, user: JwtPayload)`	`Promise<DocumentTemplate>`	Update a template.

Method	Signature	Returns	Description
`delete`	`delete(id: string, user: JwtPayload)`	`Promise<void>`	Delete a template.

Dependencies

- `PrismaService`

Where it refuses work

- `TemplateService` stops the work with `NotFoundException` when `!template`.
- `TemplateService` stops the work with `NotFoundException` when `!template.isPublic && template.organizationId !== user.organizationId`.
- `TemplateService` stops the work with `NotFoundException` when `!project` — “Project not found”.
- `TemplateService` stops the work with `NotFoundException` when `!document`.
- `TemplateService` stops the work with `BadRequestException` when `counter > 100` — “Unable to generate unique slug”.
- `TemplateService` stops the work with an early return when `error instanceof NotFoundException`, in 4 places.

When something fails

- `TemplateService` handles failure in 7 places: it lets it reach the caller in all 7.

Diagram

```
sequenceDiagram
    participant Client
    participant Controller
    participant TemplateService
    participant TemplateRepository
    participant DocumentService

    Client->>Controller: Create or use template request
    Controller->>TemplateService: create() / createDocumentFromTemplate()

    alt Create template
        TemplateService->>TemplateRepository: Persist template
        TemplateRepository-->>TemplateService: DocumentTemplate
        TemplateService-->>Controller: DocumentTemplate
    else Create document from template
        TemplateService->>TemplateRepository: getById(templateId)
        TemplateRepository-->>TemplateService: DocumentTemplate
        TemplateService->>DocumentService: Create document from template data
        DocumentService-->>TemplateService: Document
        TemplateService->>TemplateRepository: Update usage tracking
        TemplateService-->>Controller: Document
    end

    Controller-->>Client: Response
```

Usage

```
import { Injectable } from '@nestjs/common';
import { TemplateService } from './template.service';

@Injectable()
export class DocumentWorkflowService {
  constructor(private readonly templateService: TemplateService) {}

  async createFromTemplate(templateId: string, userId: string) {
    const template = await this.templateService.getById(templateId);

    const document = await this.templateService.createDocumentFromTemplate(
      template.id,
      userId,
    );

    return document;
  }

  async createReusableTemplate(documentId: string, name: string, userId: string) {
    return this.templateService.saveAsTemplate(documentId, {
      name,
      createdById: userId,
    });
  }
}
```

AI Coding Instructions

- Keep template-specific business logic in `TemplateService`; controllers should only validate request DTOs and delegate operations.
- Use `getById()` before operations that depend on template existence, and preserve the service's not-found error behavior.
- When creating a document from a template, ensure usage-tracking updates occur only after successful document creation.
- Preserve document content and metadata separation when implementing `saveAsTemplate()` so document-specific state is not copied unintentionally.
- Update related DTOs, persistence models, and controller endpoints together when adding template fields or workflow behavior.

Relationships

- DEPENDS_ON → `PrismaService`

Referenced By

- `TemplateController` (DEPENDS_ON)
- `TemplateModule` (MODULE_PROVIDES)
- `TemplateModule` (MODULE_EXPORTS)