UrlService

September 5, 2026

UrlService

Kind: Service

Source: `atloria-monorepo/apps/api/src/document/services/url.service.ts`

`UrlService` centralizes URL-safe identifier generation, slug creation, URL construction, and URL parsing for projects, collections, and documents. It provides a consistent routing contract across the backend, ensuring generated URLs can be validated and parsed back into their constituent identifiers. As a NestJS service, it can be injected into controllers and other domain services that create or resolve shareable resource URLs.

Methods

Method	Signature	Returns	Description
`generateUrlId`	`generateUrlId(maxAttempts: unknown)`	`Promise<string>`	Generate a unique URL ID for documents Uses nanoid with a URL-safe alphabet (10 characters) Includes collision detection with retry logic
`generateCollectionUrlId`	`generateCollectionUrlId(maxAttempts: unknown)`	`Promise<string>`	Generate a unique URL ID for collections Same logic as documents but checks collection table
`generateProjectUrlId`	`generateProjectUrlId(maxAttempts: unknown)`	`Promise<string>`	Generate a unique URL ID for projects Same logic as documents but

Method	Signature	Returns	Description
			checks project table
`buildProjectUrl`	`buildProjectUrl(slug: string, urlId: string)`	`string`	Build a project URL from components In production with DOCS_DOMAIN set, returns the subdomain URL.
`parseProjectUrl`	`parseProjectUrl(path: string)`	`{ slug: string; urlId: string }	null`	
`generateSlug`	`generateSlug(title: string)`	`string`	Generate an SEO-friendly slug from a title - Lowercase - Replace spaces and special chars with hyphens - Remove leading/trailing hyphens - Max 100 characters
`parseDocumentUrl`	`parseDocumentUrl(path: string)`	`ParsedDocumentUrl	null`	
`buildDocumentUrl`	`buildDocumentUrl(slug: string, urlId: string, audience: string)`	`string`	Build a document URL from components
`buildCollectionUrl`	`buildCollectionUrl(slug: string, urlId: string)`	`string`	Build a collection URL from components
`isValidUrlId`	`isValidUrlId(urlId: string)`	`boolean`	Validate a URL ID format Must be exactly 10 alphanumeric characters
`isValidSlug`	`isValidSlug(slug: string)`	`boolean`	Validate a slug format Must be lowercase, alphanumeric with hyphens, no

Method	Signature	Returns	Description
			leading/trailing hyphens
`isValidAudienceSlug`	`isValidAudienceSlug(slug: string)`	`boolean`	Validate an audience slug format Must be lowercase, alphanumeric with hyphens, max 50 chars Used for URL-based audience routing security validation
`extractUrlIdFromPath`	`extractUrlIdFromPath(path: string)`	`string	null`	

Dependencies

- `PrismaService`

Where it refuses work

- `UrlService` stops the work with an early return when `!existingDoc`.
- `UrlService` stops the work with an early return when `!existingCollection`.
- `UrlService` stops the work with an early return when `!existingProject`.
- `UrlService` stops the work with an early return when `docsDomain`.
- `UrlService` stops the work with an early return when `audience`.
- `UrlService` stops the work with an early return when `!path || path.length < 11`.

Diagram

```
sequenceDiagram
  participant Controller
  participant UrlService
  participant Database

  Controller->>UrlService: generateProjectUrlId()
  UrlService->>Database: Check URL ID uniqueness
  Database-->>UrlService: ID available
  UrlService-->>Controller: urlId

  Controller->>UrlService: generateSlug()
  UrlService-->>Controller: slug

  Controller->>UrlService: buildProjectUrl()
  UrlService-->>Controller: project URL
```

```
  Controller->>UrlService: parseProjectUrl()
  UrlService-->>Controller: { slug, urlId } | null
```

Usage

```
import { Injectable } from '@nestjs/common';
import { UrlService } from '../document/services/url.service';

@Injectable()
export class ProjectLinkService {
  constructor(private readonly urlService: UrlService) {}

  async createProjectLink() {
    const urlId = await this.urlService.generateProjectUrlId();
    const slug = this.urlService.generateSlug();

    // Build the project URL using the service's configured URL context.
    const projectUrl = this.urlService.buildProjectUrl();

    return {
      slug,
      urlId,
      projectUrl,
    };
  }

  resolveProjectLink() {
    const parsed = this.urlService.parseProjectUrl();

    if (!parsed) {
      return null;
    }

    return {
      slug: parsed.slug,
      urlId: parsed.urlId,
    };
  }
}
```

AI Coding Instructions

- Inject `UrlService` rather than duplicating slug, URL ID, parsing, or URL-building logic in controllers and domain services.
- Use the resource-specific ID generators (`generateProjectUrlId` , `generateCollectionUrlId` , and `generateUrlId`) to preserve the expected URL namespace and uniqueness behavior.
- Treat `parseProjectUrl()` and `parseDocumentUrl()` results as nullable; validate the result before accessing parsed fields.

- Use `isValidUrlId()` when accepting URL IDs from external requests before performing database lookups or resource resolution.
- Keep URL format changes centralized in this service, since routing, shared links, and persisted URL identifiers may depend on its parsing contract.

Relationships

- DEPENDS_ON → `PrismaService`

Referenced By

- `DocumentModule` (MODULE_PROVIDES)
- `DocumentModule` (MODULE_EXPORTS)
- `DocumentService` (DEPENDS_ON)
- `PublicDocumentController` (DEPENDS_ON)
- `DocAutomationService` (DEPENDS_ON)
- `ProjectService` (DEPENDS_ON)
- `PublicProjectController` (DEPENDS_ON)