ProjectDomainsService

September 4, 2026

ProjectDomainsService

Kind: Service

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

`ProjectDomainsService` manages the lifecycle of a project's custom domain, including creation, verification checks, retries, and removal. It also resolves public hostnames to published project identifiers and provides reconciliation and monitoring hooks for keeping domain state synchronized with the underlying hosting or DNS provider.

Methods

Method	Signature	Returns	Description
`get`	`get(projectId: string, user: JwtPayload)`	`Promise<CustomDomainView | null>`	
`create`	`create(projectId: string, user: JwtPayload, rawDomain: string)`	`Promise<CustomDomainView>`	
`check`	`check(projectId: string, user: JwtPayload)`	`unknown`	The interactive check: run DNS, transition state, provision the Ingress on verification, promote to live when the cert secret is populated.

Method	Signature	Returns	Description
`retry`	`retry(projectId: string, user: JwtPayload)`	`Promise<CustomDomainView>`	Retry / Recover: clear the failure and re-run the check (throttle certificate retries).
`remove`	`remove(projectId: string, user: JwtPayload)`	`Promise<{ ok: boolean; warning?: string }>`	
`resolvePublicHost`	`resolvePublicHost(host: string)`	`Promise<{ publishedSlug: string; urlId: string } | null>`	
`reconcile`	`reconcile()`	`unknown`	Hourly: promote issuing→live once certs land; heal missing Ingresses (drift).
`monitor`	`monitor()`	`unknown`	Daily: live domains whose DNS quietly broke go offline (surfaced in the UI as Recover).

Dependencies

- `PrismaService`
- `DomainDnsService`
- `K8sIngressService`
- `EmailService`

Where it refuses work

- `ProjectDomainsService` stops the work with `NotFoundException` when `!d || d.status === 'removed'` — “No custom domain configured”, in 2 places.
- `ProjectDomainsService` stops the work with `NotFoundException` when `!project` — “Project not found”.
- `ProjectDomainsService` stops the work with `BadRequestException` when `!DOMAIN_RE.test(domain)` — “Enter a valid domain, e.g. docs.yourcompany.com”.
- `ProjectDomainsService` stops the work with `BadRequestException` when `FORBIDDEN_SUFFIXES.some((s) => domain === s || domain.endsWith( .${s} ))` — “That domain belongs to the platform — use your own domain.”.
- `ProjectDomainsService` stops the work with `ConflictException` when `existing && existing.status !== 'removed'` — “This project already has a custom domain. Remove it first.”.
- `ProjectDomainsService` stops the work with `ConflictException` when `(err as { code?: string }).code === 'P2002'` — “This domain is already connected to another project.”.

When something fails

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

Diagram

```
sequenceDiagram
  participant Client
  participant Service as ProjectDomainsService
  participant Provider as Domain/DNS Provider
  participant Store as Project Domain Store

  Client->>Service: get()
  Service->>Store: Load current domain
  Store-->>Service: CustomDomainView | null
  Service-->>Client: Current domain state

  Client->>Service: create()
  Service->>Provider: Provision domain configuration
  Provider-->>Service: Provider domain details
  Service->>Store: Persist domain state
  Service-->>Client: CustomDomainView

  Client->>Service: check() / retry()
  Service->>Provider: Verify DNS and provisioning status
  Provider-->>Service: Verification result
  Service->>Store: Update domain status
```

```
  Service-->>Client: Updated status

  Client->>Service: resolvePublicHost()
  Service->>Store: Resolve hostname mapping
  Store-->>Service: publishedSlug and urlId
  Service-->>Client: Public host resolution
```

Usage

```
import { Injectable } from '@nestjs/common';
import { ProjectDomainsService } from './project-domains.service';

@Injectable()
export class ProjectDomainController {
  constructor(
    private readonly projectDomainsService: ProjectDomainsService,
  ) {}

  async provisionDomain() {
    const existing = await this.projectDomainsService.get();

    if (existing) {
      return existing;
    }

    const domain = await this.projectDomainsService.create();

    // Trigger a verification/status refresh after provisioning.
    await this.projectDomainsService.retry();

    return domain;
  }

  async resolvePublicProject() {
    const resolved = await this.projectDomainsService.resolvePublicHost();

    if (!resolved) {
      return null;
    }

    return {
      slug: resolved.publishedSlug,
      urlId: resolved.urlId,
    };
  }

  async removeDomain() {
    const result = await this.projectDomainsService.remove();

    if (!result.ok) {
```

```
        throw new Error(result.warning ?? 'Unable to remove custom domain');
      }

      return result;
    }
  }
```

AI Coding Instructions

- Use `get()` before `create()` when the caller must avoid provisioning duplicate domain configurations.
- Treat `check()`, `reconcile()`, and `monitor()` as status/synchronization operations; do not assume their return values have a stable public shape because they are typed as `unknown`.
- Use `retry()` for explicit re-verification or recovery flows after DNS records or provider-side configuration changes.
- Handle `resolvePublicHost()` returning `null`; a hostname may not yet be associated with a published project.
- Check both `ok` and the optional `warning` from `remove()` so cleanup failures can be surfaced without discarding provider-specific context.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `DomainDnsService`
- DEPENDS_ON → `K8sIngressService`
- DEPENDS_ON → `EmailService`

Referenced By

- `ProjectController` (DEPENDS_ON)
- `ProjectModule` (MODULE_PROVIDES)
- `PublicProjectController` (DEPENDS_ON)