DomainDnsService

September 4, 2026

DomainDnsService

Kind: Service

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

`DomainDnsService` is a NestJS backend service responsible for defining the DNS records required by a project domain and validating whether those records are configured correctly. It exposes record specifications for clients or provisioning workflows and provides an asynchronous DNS check result for domain-status verification.

Methods

Method	Signature	Returns	Description
`recordSpecs`	`recordSpecs(domain: string, token: string, cnameTarget: string)`	`DnsRecordSpec[]`	The two records a customer must create.
`check`	`check(domain: string, token: string, cnameTarget: string, clusterIp: string)`	`Promise<DomainDnsResult>`	Full check: records + detectors + registrar.

Where it refuses work

- `DomainDnsService` stops the work with an early return when `!ips.length`.
- `DomainDnsService` stops the work with an early return when `hit`.

When something fails

- `DomainDnsService` handles failure in 1 place: it turns it into a return value in all 1.

Diagram

```
sequenceDiagram
  participant Client as API Consumer
  participant Service as DomainDnsService
  participant DNS as DNS Provider / Resolver

  Client->>Service: recordSpecs()
  Service-->>Client: DnsRecordSpec[]

  Client->>Service: check()
  Service->>DNS: Resolve required DNS records
  DNS-->>Service: Record lookup results
  Service-->>Client: Promise<DomainDnsResult>
```

Usage

```
import { Injectable } from '@nestjs/common';
import { DomainDnsService } from './domain-dns.service';

@Injectable()
export class DomainVerificationService {
  constructor(private readonly domainDnsService: DomainDnsService) {}

  async verifyDomain() {
    const requiredRecords = this.domainDnsService.recordSpecs();
    const dnsResult = await this.domainDnsService.check();

    return {
      requiredRecords,
      verified: dnsResult,
    };
  }
}
```

AI Coding Instructions

- Keep DNS record definitions centralized in `recordSpecs()` so UI, API, and verification logic use the same requirements.
- Treat `check()` as an asynchronous external-operation boundary; handle resolver failures, timeouts, and unavailable DNS responses safely.
- Preserve the `DnsRecordSpec` and `DomainDnsResult` contracts when adding record types or validation fields.
- Inject `DomainDnsService` through NestJS dependency injection rather than constructing it directly.

- Ensure DNS validation accounts for propagation delays and does not assume a record is immediately available after configuration.

Referenced By

- `ProjectDomainsService` (DEPENDS_ON)
- `ProjectModule` (MODULE_PROVIDES)