TrialService

September 4, 2026

TrialService

Kind: Service

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

One-click public-repo trial (Phase 1.1) — the answer to DeepWiki's zero-friction entry. Paste a public GitHub URL → docs hub + grounded chat + MCP + SDK, no signup. Public repos
clone anonymously, so this needs NO GitHub App/OAuth. Guardrails: URL shape validation,
per-repo dedupe (same repo → same trial project, regenerations reused), a global daily
cap, and per-IP rate limiting at the controller. All trials live under a system org whose credits are auto-topped — normal metering still records every token for cost visibility.

`TrialService` powers the no-signup trial flow for public GitHub repositories. It creates or reuses trial projects under a system organization, coordinates documentation generation and status checks, supports claiming a trial project, and removes expired trial resources while preserving normal usage metering.

Methods

Method	Signature	Returns	Description
`startTrial`	`startTrial(repoUrl: string)`	`Promise<{	

```
projectId: string;
status: 'ready' | 'generating';
repo: string;
```

}> | | | status | status(projectId: string) | unknown | Public status — trial projects only (never leaks arbitrary project state). | | claim | claim(projectId: string, user: JwtPayload) | Promise<{ projectId: string; claimed: boolean }> | P2 "Save this project":

`re-parent a trial into the claimant's organization. |`
| `cleanupExpiredTrials` | `cleanupExpiredTrials()` | Promise` | Unclaimed trials expire after TRIAL_TTL_DAYS. |

Dependencies

- `PrismaService`
- `TechnicalDocsQueue`
- `TechnicalDocsService`
- `ProjectService`

Where it refuses work

- `TrialService` stops the work with `BadRequestException` when `!m` — “Provide a public GitHub repository URL (https://github.com/owner/repo).”.
- `TrialService` stops the work with `BadRequestException` when `today >= DAILY_GLOBAL_CAP` — “The free trial is at capacity today — please try again tomorrow or sign up.”.
- `TrialService` stops the work with `NotFoundException` when `!project` — “Trial not found.”.
- `TrialService` stops the work with `NotFoundException` when `!project` — “Trial project not found — it may already have been claimed.”.
- `TrialService` stops the work with an early return when `status.hasSnapshot`.
- `TrialService` stops the work with an early return when `!expired.length`.

When something fails

- `TrialService` handles failure in 3 places: it logs it and continues in all 3.

Diagram

```
sequenceDiagram
  participant User
  participant Controller as Trial Controller
  participant Service as TrialService
  participant DB as Database
  participant GitHub as Public GitHub Repo
  participant Pipeline as Generation Pipeline

  User->>Controller: Submit public GitHub URL
```

```
  Controller->>Controller: Validate URL and enforce IP rate limit
  Controller->>Service: startTrial()
  Service->>DB: Check global daily cap
  Service->>DB: Find existing trial by repository

  alt Existing trial project
    DB-->>Service: Existing project
    Service-->>Controller: projectId + current status
  else New trial project
    Service->>DB: Create project in system org
    Service->>GitHub: Clone repository anonymously
    Service->>Pipeline: Queue documentation generation
    Service-->>Controller: projectId + generating status
  end

  Controller-->>User: Trial project response

  User->>Controller: Check trial status
  Controller->>Service: status()
  Service-->>Controller: Current generation status

  User->>Controller: Claim trial project
  Controller->>Service: claim()
  Service->>DB: Transfer or associate project with user
  Service-->>Controller: projectId + claimed result
```

Usage

```
import { Controller, Get, Post } from '@nestjs/common';
import { TrialService } from './trial.service';

@Controller('trial')
export class TrialController {
  constructor(private readonly trialService: TrialService) {}

  @Post()
  async start() {
    // URL validation and per-IP rate limiting should happen before this call.
    const trial = await this.trialService.startTrial();

    return {
      projectId: trial.projectId,
      repo: trial.repo,
      status: trial.status,
    };
  }

  @Get('status')
  async getStatus() {
    return this.trialService.status();
```

```
  }

  @Post('claim')
  async claim() {
    const result = await this.trialService.claim();

    return {
      projectId: result.projectId,
      claimed: result.claimed,
    };
  }
}
```

AI Coding Instructions

- Keep public GitHub URL validation and per-IP rate limiting in the controller or guard layer; `TrialService` should focus on trial lifecycle orchestration.
- Preserve repository-level deduplication so repeated requests for the same repository reuse the existing trial project and generation output.
- Enforce the global daily trial cap before creating new projects, but do not block access to an already-created deduplicated trial.
- Ensure trial projects remain associated with the system organization until `claim()` successfully transfers or links them to a user-owned context.
- Keep normal token and generation metering enabled for trials, even when system-organization credits are automatically topped up.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `TechnicalDocsQueue`
- DEPENDS_ON → `TechnicalDocsService`
- DEPENDS_ON → `ProjectService`

Referenced By

- `TrialController` (DEPENDS_ON)
- `TrialModule` (MODULE_PROVIDES)