GithubAppService

September 5, 2026

GithubAppService

Kind: Service

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

Foundation F1: the Atloria GitHub App. One app, installed per customer org/personal account with per-install repo grants — GitHub enforces tenant isolation, tokens are org-owned (survive employee churn), and webhooks arrive tagged with the installation. Unlocks: one-click public-repo trial, self-serve onboarding, freshness at scale.

Env: GITHUB_APP_ID, GITHUB_APP_PRIVATE_KEY (PEM, \n-escaped), GITHUB_APP_WEBHOOK_SECRET.

`GithubAppService` encapsulates Atloria's GitHub App ("Foundation F1") integration: creating and using installation-scoped credentials, validating webhook authenticity, and interacting with GitHub on behalf of a specific customer org/user installation. It relies on GitHub's installation model to enforce tenant isolation (per-install repo grants) and to ensure tokens are org-owned (survive employee churn), enabling self-serve onboarding, public-repo trials, and scalable data freshness.

Methods

Method	Signature	Returns	Description
`installationToken`	`installationToken(installationId: number	`bigint)`	`Promise<string>`
`cloneCredentialsFor`	`cloneCredentialsFor(repoUrl: string)`	`Promise<{ username: string; password: string }	null>`
`createCheckRun`	`createCheckRun(repoFullName: string, headSha: string, opts: { name: string; conclusion?: string; detailsUrl?: string; summary?: string })`	`Promise<void>`	B2: create a Check Run on a head commit carrying the docs-preview link (details_url).

Method	Signature	Returns	Description
`postIssueComment`	`postIssueComment(repoFullName: string, prNumber: number, body: string)`	`Promise<void>`	B2: post a single issue/PR comment (issues API — PRs are issues).
`verifySignature`	`verifySignature(signature: string	undefined, rawBody: string	Buffer
`handleInstallationEvent`	`handleInstallationEvent(action: string, payload: any)`	`Promise<void>`	installation / installation_repositories events → upsert the install registry.

Dependencies

- `ConfigService`
- `PrismaService`

Where it refuses work

- `GithubAppService` stops the work with `Error` when `!this.appId || !this.privateKey` — “GitHub App not configured”.
- `GithubAppService` stops the work with `Error` when `!creds?.password`.
- `GithubAppService` stops the work with `ForbiddenException` when `process.env.NODE_ENV === 'production'` — “GitHub App webhook secret is not configured.”.
- `GithubAppService` stops the work with `ForbiddenException` when `!signature || !rawBody` — “Missing webhook signature”.
- `GithubAppService` stops the work with `ForbiddenException` when `a.length !== b.length || !crypto.timingSafeEqual(a, b)` — “Invalid webhook signature”.
- `GithubAppService` stops the work with an early return when `!this.configured`.

When something fails

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

Diagram

```
sequenceDiagram
  autonumber
  participant GH as GitHub
  participant API as Atloria API (Webhook Controller)
  participant GAS as GithubAppService
```

```
  participant GHA as GitHub API

  GH->>API: Webhook event (X-Hub-Signature-256, installation.id, payload)
  API->>GAS: verifyWebhookSignature(rawBody, headers)
  GAS-->>API: ok / throw (invalid signature)

  API->>GAS: getInstallationClient(installationId)
  GAS->>GH: Create JWT (app id + private key)
  GAS->>GHA: POST /app/installations/{id}/access_tokens (JWT)
  GHA-->>GAS: installation access token (scoped to granted repos)
  GAS-->>API: Octokit client (auth = installation token)

  API->>GHA: GitHub API calls via client (repos, pulls, commits, etc.)
  GHA-->>API: responses (tenant-isolated by installation grants)
```

Usage

```
import { Injectable } from '@nestjs/common';
import type { Request } from 'express';
import { GithubAppService } from './github-app.service';

@Injectable()
export class GithubWebhookHandler {
  constructor(private readonly githubApp: GithubAppService) {}

  async handle(req: Request) {
    // IMPORTANT: signature verification typically needs the raw request body
    // Ensure your Nest/Express setup preserves rawBody for webhook routes.
    await this.githubApp.verifyWebhookSignature(
      // @ts-expect-error depends on your raw body middleware
      req.rawBody ?? Buffer.from(JSON.stringify(req.body)),
      req.headers,
    );

    const installationId = req.body?.installation?.id;
    if (!installationId) throw new Error('Missing installation.id in webhook payload');

    const octokit = await this.githubApp.getInstallationClient(installationId);

    // Example: list repos accessible to this installation (tenant-scoped)
    const { data } = await octokit.apps.listReposAccessibleToInstallation();
    return data.repositories.map(r => ({ id: r.id, full_name: r.full_name }));
  }
}
```

AI Coding Instructions

- Always operate “per installation”: require `installation.id` (from webhook payload or stored mapping) and use installation-scoped tokens/clients; never use a user PAT for backend GitHub operations.

- Webhook verification must use the exact raw bytes of the request body (before JSON parsing/mutation) with `GITHUB_APP_WEBHOOK_SECRET`; failing to preserve `rawBody` is a common pitfall.
- `GITHUB_APP_PRIVATE_KEY` is PEM with `\n`-escaped newlines—normalize it (replace `\\n` with `\n`) before signing JWTs, or token creation will fail.
- Treat installation access tokens as short-lived; do not persist them long-term—recreate via the app JWT flow when needed and handle 401/403 by refreshing.
- Keep tenant isolation intact: never broaden repo access in code; rely on GitHub's per-installation grants and ensure all queries are made through the installation-authenticated Octokit instance.

Relationships

- DEPENDS_ON → `configservice`
- DEPENDS_ON → `PrismaService`

Referenced By

- `DocsPrService` (DEPENDS_ON)
- `GithubAppController` (DEPENDS_ON)
- `GithubAppModule` (MODULE_PROVIDES)
- `GithubAppModule` (MODULE_EXPORTS)
- `WebhookService` (DEPENDS_ON)