GithubAppController

September 4, 2026

GithubAppController

Kind: Controller

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

GitHub App webhook receiver (Foundation F1). Handles installation lifecycle (the install
registry) and push events (delegates to the existing push pipeline: staleness fan-out + optional auto-regen trigger). Verified by the APP's HMAC secret — one secret for all customers, unlike per-project manual webhooks.

`GithubAppController` is the NestJS webhook receiver for the Atloria GitHub App (Foundation F1). It verifies incoming webhook requests using the GitHub App HMAC secret, manages the installation lifecycle via the install registry, and handles `push` events by delegating into the existing push pipeline (staleness fan-out and optional auto-regeneration triggers). This centralizes webhook handling under a single shared secret (per app) instead of per-project manual webhook secrets.

Diagram

```
graph LR
  GH[GitHub App Webhooks] -->|HTTP POST + HMAC signature| C[GithubAppController]
  C --> V[HMAC Verification]
  V -->|invalid| R401[Reject 401/403]
  V -->|valid| D{Event type}
  D -->|installation / lifecycle| IR[Install Registry]
  D -->|push| PP[Push Pipeline]
  PP --> SF[Staleness Fan-out]
  PP --> AR[Optional Auto-Regen Trigger]
  IR --> OK[200 OK]
  AR --> OK
  SF --> OK
```

Usage

```ts
import { Controller, Post, Headers, Body, Req, Res } from '@nestjs/common';
import type { Request, Response } from 'express';

// This shows how GithubAppController is typically used: it's mounted as a NestJS controller
// and receives GitHub App webhook POSTs. The verification + routing is handled inside.
@Controller('/github-app')
export class GithubAppController {
  @Post('/webhook')
  async webhook(
    @Headers('x-github-event') event: string,
    @Headers('x-hub-signature-256') signature: string,
    @Body() payload: any,
    @Req() req: Request,
    @Res() res: Response,
  ) {
    // Internally (in the real implementation):
    // 1) Verify signature using the GitHub App webhook secret
    // 2) If installation event: update install registry
    // 3) If push event: delegate to push pipeline (staleness fan-out, optional auto-regen)
    // 4) Return 2xx quickly to GitHub
    return res.status(200).send({ ok: true, event });
  }
}

// Example HTTP call (for local testing) using fetch:
async function sendTestWebhook() {
  await fetch('http://localhost:3000/github-app/webhook', {
    method: 'POST',
    headers: {
      'content-type': 'application/json',
      'x-github-event': 'push',
      // In real GitHub delivery this is computed HMAC SHA-256 of the raw request body:
      'x-hub-signature-256': 'sha256=...',
    },
    body: JSON.stringify({ ref: 'refs/heads/main', repository: { full_name: 'acme/repo' } }),
  });
}
```

AI Coding Instructions

- Preserve **raw request body** access for HMAC validation; don't JSON-transform/pretty-print before computing the signature or verification will fail.
- Keep webhook handlers **fast and idempotent**: acknowledge GitHub with 2xx quickly and push heavy work into the existing push pipeline/queues.
- Route by GitHub headers (`x-github-event` , delivery id) and ensure installation lifecycle events update the **install registry** consistently (handle install, uninstall, suspend/unsuspend).

- Treat the webhook secret as **app-wide** (not per project); avoid introducing per-customer secret branching unless the architecture changes explicitly.
- When adding new event types, follow the existing delegation pattern: controller validates + normalizes → service/pipeline performs domain work (staleness fan-out, auto-regen, etc.).

Relationships

- MODULE_DECLARES → `handle`
- DEPENDS_ON → `GithubAppService`
- DEPENDS_ON → `WebhookService`
- DEPENDS_ON → `DocsPrTrigger`
- DEPENDS_ON → `GitSyncTrigger`

Referenced By

- `GithubAppModule` (MODULE_DECLARES)