Toggly NestJS

HTTP modules · request-scoped evaluation · FeatureFlagGuard

Install

```
npm i @ops-ai/toggly-nestjs \
  reflect-metadata rxjs
```

Backend App Key only — never a frontend or management credential. Trusted `definitions-signed`, not evaluated-signed.

Configure once

```
TogglyModule.forRoot({
  appKey: process.env.TOGGLY_APP_KEY,
  environment: process.env.TOGGLY_ENVIRONMENT
    ?? 'Production',
  verifySignatures: true,
  featureDefaults: { 'new-dashboard': false },
  contextFactory: (request) => ({
    identity: request.user?.id ?? 'anonymous',
    groups: request.user?.groups,
    claims: request.user?.claims,
  }),
})
```

One root per app. `forRootAsync` for ConfigModule. `isGlobal: true` to export. Missing identity becomes `anonymous`. Never `provider.client.setIdentity` from HTTP handlers.

Programmatic

```
await this.toggly.isFeatureOn('new-dashboard')
await this.toggly.isFeatureOff('maintenance')
await this.toggly.evaluateFeatureGate(
  ['a', 'b'], 'any', false)
```

Empty programmatic gates allow. Missing flags use `featureDefaults` or false.

Entities

```
await toggly.isFeatureOn('ExpressCheckout', {
  entity: {
    kind: 'Order', key: 'ord-vip',
    attributes: { Vip: true, Total: 199 },
  },
})
await toggly.isFeatureOn('beta-access', {
  context: { identity: 'alice' },
})
```

`overrides.context` replaces top-level fields for that call; it never changes ambient identity. Register `provider.client.registerContext('Order', …)` for domain objects.

Lifecycle

```
app.enableShutdownHooks();
await provider.client.refresh();
await this.toggly.recordUsage(key);
provider.client.flushTelemetry();
```

Call `app.close()` so core flushes telemetry and releases sockets. Do not inject request-scoped services into singleton gateways.

Reliability

- `initialized` means startup finished, including defaults/cache fallback — check `state.error`.
- Refresh failure is not a thrown evaluation error; guards keep last-known-good defs.
- Durable file cache is trusted server state, not a cryptographic trust boundary.

Toggly NestJS

HTTP modules · request-scoped evaluation · FeatureFlagGuard

Guards

```
@Controller('dashboard')
export class DashboardController {
  constructor(
    @Inject(TogglyService)
    private readonly toggly: TogglyService) {}

  @Get()
  @UseGuards(FeatureFlagGuard)
  @FeatureFlag(['new-dashboard', 'api-v2'],
    { requirement: 'any' })
  show(@FeatureEnabled('enhanced-submit') on: boolean)
    return { enhancedSubmit: on };
  }

  @Get('legacy')
  @UseGuards(FeatureFlagGuard)
  @FeatureFlag('api-v2', { negate: true })
  legacy() { return { version: 1 }; }
}
```

Disabled defaults to **404**; `{ disabledStatus: 403 }` for 403. Thrown evaluation errors yield **503**. HTTP only — not GraphQL/WebSocket/jobs.

Pitfalls

- Gates complement auth — they do not replace it.
- Distinct anonymous visitors need distinct identities for sticky rollouts.
- Return `request.userAgent`, `acceptLanguage`, and `country` from a trusted source for HTTP segment filters.
- No Terminus indicator, GraphQL context, or job integration in this adapter.

Resources

- Docs: `docs.toggly.io/sdks/nestjs`
- Node core sheet: `/cheatsheets/nodejs`
- Sample: `Toggly.Samples/nestjs-sdk`