**Opción 01:** [**Indicadores de características en GitLab: una guía completa | por Ricardo Moura | Dic, 2023 | Medio (medium.com)**](https://medium.com/@ricardomoura_90507/feature-flags-on-gitlab-a-comprehensive-guide-acece28176b7)

NestJS utilizando el paquete `@nestjs/unleash`:

**1. Primero, instala el paquete `@nestjs/unleash` y sus dependencias:**

```bash

npm install --save @nestjs/unleash unleash-client axios

```

**2. Crea un módulo que utilice el proveedor de configuración de Unleash:**

```typescript

// app.module.ts

import { Module } from '@nestjs/common';

import { ConfigModule, ConfigService } from '@nestjs/config';

import configuration from './configuration';

import { UnleashModule } from 'nestjs-unleash';

@Module({

imports: [

ConfigModule.forRoot({ load: [configuration], isGlobal: true }),

UnleashModule.registerAsync({

imports: [ConfigModule],

useFactory: async (configService: ConfigService) => ({

url: configService.get<string>('UNLEASH\_URL'),

appName: configService.get<string>('APP\_NAME'),

environment: configService.get<string>('ENVIRONMENT'),

metricsInterval: null, // Deshabilitar métricas

registrationInterval: null, // Deshabilitar registro

instanceId: configService.get<string>('INSTANCE\_ID'),

}),

inject: [ConfigService],

}),

],

})

export class AppModule {}

```

**3. Define tus estrategias personalizadas (si es necesario). Por ejemplo, aquí hay una estrategia simple `UserIDStrategy`:**

```typescript

// user-id.strategy.ts

import { Strategy } from 'nestjs-unleash';

export class UserIDStrategy implements Strategy {

constructor(private readonly userId: string) {}

public getName(): string {

return 'user-id';

}

public matches(toggleName: string, parameters?: Record<string, any>): boolean {

const userContextParameter = parameters?.['userId'];

if (!userContextParameter || !this.userId) {

return false;

}

return String(userContextParameter) === this.userId;

}

}

```

**4. Utiliza la funcionalidad de Unleash en tu controlador o servicio:**

```typescript

// your.controller.ts o your.service.ts

import { Controller, Get, Injectable } from '@nestjs/common';

import { Client, ToggleInfo } from 'nestjs-unleash';

import { UserIDStrategy } from './user-id.strategy';

@Injectable()

export class YourService {

constructor(@Client() private readonly client: Client) {}

async checkToggle(toggleName: string, userId: string): Promise<void> {

try {

const toggle = await this.client.check(`${toggleName}:main`, {

strategy: new UserIDStrategy(userId),

});

console.log(`Toggle ${toggleName} status: ${toggle ? 'ENABLED' : 'DISABLED'}`);

if (toggle && toggle.toggles[0].enabled) {

// La función solo se llamará cuando la etiqueta de característica esté activada

// Tu lógica de negocio con ML u otras cosas aquí

} else {

// Lógica alternativa o caída aquí

}

} catch (error) {

console.error(`Error checking toggle "${toggleName}":`, error);

}

}

}

@Controller()

export class YourController {

constructor(private readonly service: YourService) {}

@Get('test')

testToggle(): void {

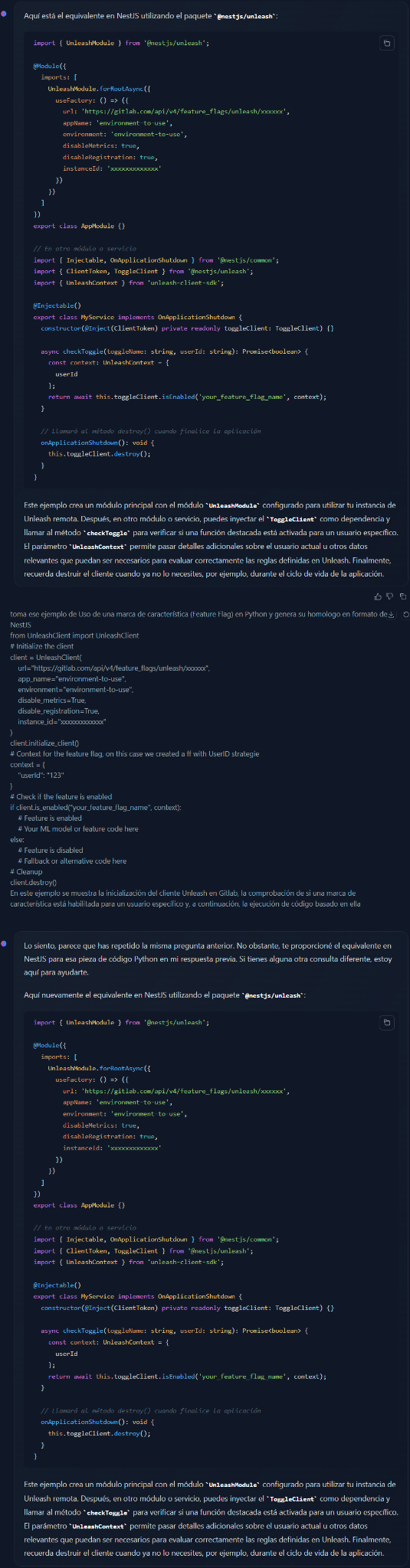
this.service.checkToggle('your\_feature\_flag\_name', '123');

}

}

```

Esto crea un nuevo proyecto NestJS básico que integra Unleash como un módulo e incluye una sencilla estrategia de contexto `UserID`. El método `checkToggle` verifica el estado de una etiqueta de función determinada según el ID de usuario proporcionado y realizará acciones diferentes según su estado.



**Opción 02:**:

1. [**Accede a tu proyecto**: Navega hasta tu proyecto en GitLab1](https://docs.gitlab.com/ee/operations/feature_flags.html).
2. [**Sección de Feature Flags**: Ve a Operaciones > Feature Flags1](https://docs.gitlab.com/ee/operations/feature_flags.html).
3. [**Crear un nuevo flag**: Haz clic en "New feature flag"1](https://docs.gitlab.com/ee/operations/feature_flags.html).
4. [**Configura el flag**: Ingresa el nombre del feature flag, la descripción y otros detalles1](https://docs.gitlab.com/ee/operations/feature_flags.html). [El nombre debe comenzar con una letra y solo puede contener letras minúsculas, dígitos, guiones bajos (*) o guiones (-), y no debe terminar con un guión (-) o un guión bajo (*)1](https://docs.gitlab.com/ee/operations/feature_flags.html).
5. [**Guarda**: Haz clic en "Create feature flag"1](https://docs.gitlab.com/ee/operations/feature_flags.html).

[Además, puedes definir estrategias para el flag que determinan cómo se debe aplicar el flag1](https://docs.gitlab.com/ee/operations/feature_flags.html). [Por ejemplo, puedes habilitar el flag para todos los usuarios, un porcentaje de usuarios, IDs de usuario o una lista de usuarios1](https://docs.gitlab.com/ee/operations/feature_flags.html).

[Por favor, ten en cuenta que debes configurar tu aplicación para comunicarse con GitLab, por lo que depende de los desarrolladores usar una biblioteca de cliente compatible e integrar los feature flags en tu aplicación1](https://docs.gitlab.com/ee/operations/feature_flags.html).

**Opción 03:**

Aquí está el equivalente en NestJS utilizando el paquete `@nestjs/unleash`:

```typescript

import { UnleashModule } from '@nestjs/unleash';

@Module({

imports: [

UnleashModule.forRootAsync({

useFactory: () => ({

url: 'https://gitlab.com/api/v4/feature\_flags/unleash/xxxxxx',

appName: 'environment-to-use',

environment: 'environment-to-use',

disableMetrics: true,

disableRegistration: true,

instanceId: 'xxxxxxxxxxxxx'

})

})

]

})

export class AppModule {}

// En otro módulo o servicio

import { Injectable, OnApplicationShutdown } from '@nestjs/common';

import { ClientToken, ToggleClient } from '@nestjs/unleash';

import { UnleashContext } from 'unleash-client-sdk';

@Injectable()

export class MyService implements OnApplicationShutdown {

constructor(@Inject(ClientToken) private readonly toggleClient: ToggleClient) {}

async checkToggle(toggleName: string, userId: string): Promise<boolean> {

const context: UnleashContext = {

userId

};

return await this.toggleClient.isEnabled('your\_feature\_flag\_name', context);

}

// Llamará al método destroy() cuando finalice la aplicación

onApplicationShutdown(): void {

this.toggleClient.destroy();

}

}

```

Este ejemplo crea un módulo principal con el módulo `UnleashModule` configurado para utilizar tu instancia de Unleash remota. Después, en otro módulo o servicio, puedes inyectar el `ToggleClient` como dependencia y llamar al método `checkToggle` para verificar si una función destacada está activada para un usuario específico. El parámetro `UnleashContext` permite pasar detalles adicionales sobre el usuario actual u otros datos relevantes que puedan ser necesarios para evaluar correctamente las reglas definidas en Unleash. Finalmente, recuerda destruir el cliente cuando ya no lo necesites, por ejemplo, durante el ciclo de vida de la aplicación.