RateLimitGuard

September 4, 2026

RateLimitGuard

Kind: Service

Source: `atloria-monorepo/apps/api/src/ai/guards/rate-limit.guard.ts`

`RateLimitGuard` is a NestJS guard that protects AI-related API endpoints by checking whether an incoming request is within its allowed rate limit before the request handler executes. It integrates into NestJS's guard lifecycle through `canActivate()` and releases any rate-limit-related resources when the module shuts down through `onModuleDestroy()`.

Methods

Method	Signature	Returns	Description
`canActivate`	`canActivate(context: ExecutionContext)`	`Promise<boolean>`	
`onModuleDestroy`	`onModuleDestroy()`	`unknown`	Cleanup Redis connection on module destroy

Dependencies

- `ConfigService`

Where it refuses work

- `RateLimitGuard` stops the work with an early return when `!userId`.
- `RateLimitGuard` stops the work with an early return when `error instanceof HttpException`.

When something fails

- `RateLimitGuard` handles failure in 2 places: it lets it reach the caller in 1, and discards it silently in 1. A failure discarded silently leaves no trace for whoever debugs this later.

Diagram

```
sequenceDiagram
    participant Client
    participant Controller
    participant Guard as RateLimitGuard
    participant Limiter as Rate Limit Store
    participant Handler as AI Endpoint Handler

    Client->>Controller: HTTP request
    Controller->>Guard: canActivate(context)
    Guard->>Limiter: Check/increment request allowance

    alt Request is within limit
        Limiter-->>Guard: Allowed
        Guard-->>Controller: true
        Controller->>Handler: Execute endpoint
        Handler-->>Client: Response
    else Rate limit exceeded
        Limiter-->>Guard: Rejected
        Guard-->>Controller: Throw/return false
        Controller-->>Client: Rate limit response
    end
```

Usage

```
import { Controller, Get, UseGuards } from '@nestjs/common';
import { RateLimitGuard } from '../ai/guards/rate-limit.guard';

@Controller('ai')
@UseGuards(RateLimitGuard)
export class AiController {
  @Get('status')
  getStatus() {
    return {
      status: 'available',
    };
  }
}
```

AI Coding Instructions

- Apply `RateLimitGuard` with `@UseGuards()` at the controller or route level for AI endpoints that need request throttling.
- Keep rate-limit decisions inside `canActivate()` so protected handlers are not executed when a request exceeds its allowance.
- Preserve NestJS guard behavior: return `true` only when access should continue, and use the established error-handling pattern when denying requests.
- Do not manually call `onModuleDestroy()`; NestJS invokes it during application shutdown to clean up guard resources.
- When adding new AI routes, ensure their rate-limit scope and client identity strategy match the existing guard configuration.

Relationships

- DEPENDS_ON → `configservice`

Referenced By

- `AIModule` (MODULE_PROVIDES)