OptionalJwtAuthGuard

September 4, 2026

OptionalJwtAuthGuard

Kind: Service

Source: `atloria-monorepo/apps/api/src/auth/guards/optional-jwt-auth.guard.ts`

Optional JWT Auth Guard

This guard attempts to authenticate the user via JWT but doesn't
require authentication. If a valid JWT is present, req.user will
be populated. If not, the request continues with req.user undefined.

Use this for public endpoints that need optional authentication
(e.g., public document viewer that shows more content for authenticated users).

`OptionalJwtAuthGuard` is a NestJS guard that attempts to authenticate incoming requests using a JWT without enforcing authentication. If a valid token is present, it attaches the decoded user to `req.user`; otherwise, the request proceeds normally with `req.user` remaining `undefined`. Use it for public endpoints that can enhance responses when a user is authenticated.

Methods

Method	Signature	Returns
`handleRequest`	`handleRequest(err: any, user: TUser)`	`TUser
`canActivate`	`canActivate(context: ExecutionContext)`	`unknown`

Where it refuses work

- `OptionalJwtAuthGuard` stops the work with an early return when `err || !user`.

Diagram

```
sequenceDiagram
  autonumber
  participant C as Client
  participant N as NestJS Pipeline
  participant G as OptionalJwtAuthGuard
  participant J as Jwt Strategy / Auth Service
  participant R as Controller/Route Handler

  C->>N: HTTP Request (may include Authorization: Bearer <jwt>)
  N->>G: canActivate(context)
  G->>J: Validate JWT (if present)
  alt JWT valid
    J-->>G: user payload
    G-->>N: allow; set req.user = payload
  else No JWT or invalid JWT
    J-->>G: validation fails / no token
    G-->>N: allow; req.user remains undefined
  end
  N->>R: Invoke handler
  R-->>C: Response (optionally tailored by req.user)
```

Usage

```
import { Controller, Get, Req, UseGuards } from '@nestjs/common';
import { Request } from 'express';
import { OptionalJwtAuthGuard } from './auth/guards/optional-jwt-auth.guard';

@Controller('documents')
export class DocumentsController {
  @Get(':id/view')
  @UseGuards(OptionalJwtAuthGuard)
  viewDocument(@Req() req: Request) {
    // If authenticated: req.user is populated by the JWT strategy
    const user = (req as any).user;

    if (user) {
      return { mode: 'enhanced', message: `Hello ${user.email ?? 'user'}`, content: 'Full conte
    }

    // If not authenticated: still allowed, but limited response
    return { mode: 'public', message: 'Hello guest', content: 'Preview content' };
  }
}
```

AI Coding Instructions

- Keep the guard **non-blocking**: it should never throw/deny solely due to missing or invalid JWT—always allow the request to continue.
- Ensure downstream code treats `req.user` as **optional** (`undefined`), and branch logic accordingly (avoid assuming authenticated state).
- Integrate with the existing **JWT strategy/passport setup** (e.g., the same validation logic as the required JWT guard), but swallow auth failures instead of propagating them.
- When adding new endpoints, use this guard for **public routes with optional personalization**, not for anything that must enforce authorization checks.

Referenced By

- `ReaderAccountModule` (MODULE_PROVIDES)