**📁 common/interceptors/ Directory Overview**

In NestJS, **interceptors** are classes that implement the NestInterceptor interfaceThey can bind extra logic before or after method execution, transform the result returned from a function, extend basic function behavior, or even override a function depending on specific conditionsOrganizing these interceptors within the common/interceptors/ directory promotes reusability and maintainability across your application

**🗂️ Suggested Subdirectories within interceptors/**

To maintain a clean and organized structure, consider categorizing your interceptors based on their functionalit: 

common/

└── interceptors/

├── logging/ # Interceptors related to logging

│ └── logging.interceptor.ts # Interceptor for logging requests and responses

├── transformation/ # Interceptors for transforming responses

│ └── transform.interceptor.ts

├── caching/ # Interceptors handling caching mechanisms

│ └── cache.interceptor.ts

└── timeout/ # Interceptors managing request timeouts

└── timeout.interceptor.ts



**📄 Detailed Breakdown of Each Subdirectory**

**1. logging/ Subdirectory**

* \**Purpose*: Contains interceptors that log details about incoming requests and outgoing responss.
* \**Example*: logging.interceptor.s 

import { Injectable, NestInterceptor, ExecutionContext, CallHandler } from '@nestjs/common';

import { Observable, tap } from 'rxjs';

@Injectable()

export class LoggingInterceptor implements NestInterceptor {

intercept(context: ExecutionContext, next: CallHandler): Observable<any> {

const request = context.switchToHttp().getRequest();

const method = request.method;

const url = request.url;

const now = Date.now();

console.log(`Incoming Request: ${method} ${url}`);

return next

.handle()

.pipe(

tap(() => console.log(`Response for ${method} ${url} - ${Date.now() - now}ms`)),

);

}

}

``



\*\*Usage\*\*:

Apply this interceptor globally or to specific controllers/methods using the `@UseInterceptors()` decoratr.

---

### 2. \*\*`transformation/` Subdirectory\*\*

- \*\*Purpose\*: Houses interceptors that modify or transform the response before it's sent to the cliet.

- \*\*Example\*: `transform.interceptor.s`



```typescript

import { Injectable, NestInterceptor, ExecutionContext, CallHandler } from '@nestjs/common';

import { Observable, map } from 'rxjs';

export interface Response<T> {

data: T;

}

@Injectable()

export class TransformInterceptor<T> implements NestInterceptor<T, Response<T>> {

intercept(context: ExecutionContext, next: CallHandler): Observable<Response<T>> {

return next.handle().pipe(map(data => ({ data })));

}

}

``



\*\*Usage\*\*:

This interceptor wraps the response data in a consistent structure, which can be beneficial for client-side handlig.

---

### 3. \*\*`caching/` Subdirectory\*\*

- \*\*Purpose\*: Contains interceptors that implement caching strategies to enhance performane.

- \*\*Example\*: `cache.interceptor.s`



```typescript

import { Injectable, NestInterceptor, ExecutionContext, CallHandler } from '@nestjs/common';

import { Observable, of } from 'rxjs';

@Injectable()

export class CacheInterceptor implements NestInterceptor {

intercept(context: ExecutionContext, next: CallHandler): Observable<any> {

const isCached = false; // Replace with actual caching logic

if (isCached) {

return of([]); // Return cached response

}

return next.handle();

}

}

``



\*\*Usage\*\*:

Implement caching logic within the interceptor to return cached responses when available, reducing unnecessary processig.

---

### 4. \*\*`timeout/` Subdirectory\*\*

- \*\*Purpose\*: Manages request timeouts to prevent long-running requests from hanging indefinitey.

- \*\*Example\*: `timeout.interceptor.s`



```typescript

import { Injectable, NestInterceptor, ExecutionContext, CallHandler, RequestTimeoutException } from '@nestjs/common';

import { Observable, throwError, TimeoutError } from 'rxjs';

import { catchError, timeout } from 'rxjs/operators';

@Injectable()

export class TimeoutInterceptor implements NestInterceptor {

intercept(context: ExecutionContext, next: CallHandler): Observable<any> {

return next.handle().pipe(

timeout(5000), // Set timeout duration

catchError(err => {

if (err instanceof TimeoutError) {

return throwError(() => new RequestTimeoutException());

}

return throwError(() => err);

}),

);

}

}

``



\*\*Usage\*\*:

Apply this interceptor to routes where you want to enforce a maximum processing time, enhancing reliabiliy.

---

## ✅ Benefits of Organizing Interceptors in `common/interceptors/`

- \*\*Centralized Logic\*: Having all interceptors in a dedicated directory promotes a single source of truth for cross-cutting concers.

- \*\*Reusability\*: Interceptors can be easily applied across different parts of the application, reducing code duplicatin.

- \*\*Maintainability\*: A well-structured directory makes it easier to manage and update interceptors as the application evolvs.

- \*\*Enhanced Readability\*: Categorizing interceptors by functionality allows developers to quickly locate and understand their purpoe.

--

By structuring your `common/interceptors/` directory as outlined above, you ensure that your application's cross-cutting concerns are handled efficiently and consistently, leading to a more robust and maintainable codebae.

If you need further assistance or examples on implementing these components, feel free to ask!

**📌 Applying Interceptors in NestJS**

Interceptors in NestJS can be applied at various levels, each serving different scopes and purposes

**1. Method-Level Interceptors**

Apply an interceptor to a specific route handler using the @UseInterceptors() decorator 

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

import { LoggingInterceptor } from './common/interceptors/logging/logging.interceptor';

@Controller('cats')

export class CatsController {

@Get()

@UseInterceptors(LoggingInterceptor)

findAll() {

return 'This action returns all cats';

}

}

```

This approach ensures that the `LoggingInterceptor` is invoked only for the `findAll` method

### 2. \*\*Controller-Level Interceptors\*\*

Apply an interceptor to all route handlers within a controller



```typescript

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

import { LoggingInterceptor } from './common/interceptors/logging/logging.interceptor';

@UseInterceptors(LoggingInterceptor)

@Controller('cats')

export class CatsController {

// All methods here will use LoggingInterceptor

}

```

This ensures that every method within `CatsController` utilizes the `LoggingInterceptor`

### 3. \*\*Global Interceptors\*\*

Apply an interceptor across the entire application

- \*\*Without Dependencies\*\*:Register in `main.ts`



```typescript

import { NestFactory } from '@nestjs/core';

import { AppModule } from './app.module';

import { LoggingInterceptor } from './common/interceptors/logging/logging.interceptor';

async function bootstrap() {

const app = await NestFactory.create(AppModule);

app.useGlobalInterceptors(new LoggingInterceptor());

await app.listen(3000);

}

bootstrap();

```

- \*\*With Dependencies\*\*:Register using the `APP\_INTERCEPTOR` token in a module



```typescript

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

import { APP\_INTERCEPTOR } from '@nestjs/core';

import { LoggingInterceptor } from './common/interceptors/logging/logging.interceptor';

@Module({

providers: [

{

provide: APP\_INTERCEPTOR,

useClass: LoggingInterceptor,

},

],

})

export class AppModule {}

```

This method allows the interceptor to leverage dependency injection, making it suitable for interceptors that depend on other services citeturn0search0

---

## 📁 Organizing Interceptors in Your Project Structure

To maintain a clean and scalable codebase, it's advisable to organize interceptors within the `common/interceptors/` directory, categorizing them based on their functionalit:



```bash

common/

└── interceptors/

├── logging/

│ └── logging.interceptor.ts

├── transformation/

│ └── transform.interceptor.ts

├── caching/

│ └── cache.interceptor.ts

└── timeout/

└── timeout.interceptor.ts

``



This structure promotes reusability and clarity, making it easier to manage and apply interceptors across different modules and controller.

---

## ✅ Best Practices

- \*\*Scope Appropriately\*\* Apply interceptors at the most appropriate level (method, controller, or global) based on their intended effec.

- \*\*Leverage Dependency Injection\*\* When interceptors depend on other services, register them using the `APP\_INTERCEPTOR` token within a module to enable dependency injectio.

- \*\*Avoid Overuse\*\* Be cautious when applying multiple global interceptors, as they can introduce complexity and affect performanc.

- \*\*Maintain Modularity\*\* Keep interceptors modular and focused on a single responsibility to enhance maintainabilit.

---