**📁 common/filters/ Directory Overview**

In NestJS, **exception filters** are specialized classes that handle errors thrown during the execution of your applicationBy organizing these filters within the common/filters/ directory, you centralize error handling logic, promoting cleaner and more maintainable code

**🗂️ Suggested Subdirectories within filters/**

To maintain an organized structure, consider categorizing your filters based on their scope or functionalit: 

common/

└── filters/

├── global/ # Filters applied across the entire application

│ └── http-exception.filter.ts

├── controller/ # Filters specific to certain controllers

│ └── user-exception.filter.ts

└── method/ # Filters specific to individual methods

└── validation-exception.filter.ts



**📄 Detailed Breakdown of Each Subdirectory**

**1. global/ Subdirectory**

* \**Purpose*: Contains filters that handle exceptions globally across the applicatin.
* \**Example*: http-exception.filter.s 

import { ExceptionFilter, Catch, ArgumentsHost, HttpException, HttpStatus } from '@nestjs/common';

import { Request, Response } from 'express';

/\*\*

\* Global exception filter to handle all HTTP exceptions.

\*/

@Catch()

export class HttpExceptionFilter implements ExceptionFilter {

catch(exception: unknown, host: ArgumentsHost) {

const ctx = host.switchToHttp();

const response = ctx.getResponse<Response>();

const request = ctx.getRequest<Request>();

const status = exception instanceof HttpException

? exception.getStatus()

: HttpStatus.INTERNAL\_SERVER\_ERROR;

const message = exception instanceof HttpException

? exception.getResponse()

: 'Internal server error';

response.status(status).json({

statusCode: status,

message,

timestamp: new Date().toISOString(),

path: request.url,

});

}

}

``



\*\*Usage\*\*:

Register this filter globally in your `main.ts` fie:



```typescript

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

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

import { HttpExceptionFilter } from './common/filters/global/http-exception.filter';

async function bootstrap() {

const app = await NestFactory.create(AppModule);

app.useGlobalFilters(new HttpExceptionFilter());

await app.listen(3000);

}

bootstrap();

``



With this setup, all unhandled exceptions in your application will be caught and processed by the `HttpExceptionFilter`, ensuring consistent error responss.

---

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

- \*\*Purpose\*: Houses filters that are specific to certain controllers, allowing for tailored exception handling within specific moduls.

- \*\*Example\*: `user-exception.filter.s`



```typescript

import { ExceptionFilter, Catch, ArgumentsHost, NotFoundException } from '@nestjs/common';

import { Request, Response } from 'express';

/\*\*

\* Controller-specific exception filter for user-related operations.

\*/

@Catch(NotFoundException)

export class UserExceptionFilter implements ExceptionFilter {

catch(exception: NotFoundException, host: ArgumentsHost) {

const ctx = host.switchToHttp();

const response = ctx.getResponse<Response>();

const request = ctx.getRequest<Request>();

response.status(404).json({

statusCode: 404,

message: 'User not found',

timestamp: new Date().toISOString(),

path: request.url,

});

}

}

``



\*\*Usage\*\*:

Apply this filter to a specific controllr:



```typescript

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

import { UserExceptionFilter } from '../common/filters/controller/user-exception.filter';

@Controller('users')

@UseFilters(UserExceptionFilter)

export class UsersController {

// Controller methods

}

``



This ensures that any `NotFoundException` thrown within the `UsersController` is handled by the `UserExceptionFilte`.

---

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

- \*\*Purpose\*: Contains filters that are specific to individual methods, providing fine-grained control over exception handlig.

- \*\*Example\*: `validation-exception.filter.s`



```typescript

import { ExceptionFilter, Catch, ArgumentsHost, BadRequestException } from '@nestjs/common';

import { Request, Response } from 'express';

/\*\*

\* Method-specific exception filter for validation errors.

\*/

@Catch(BadRequestException)

export class ValidationExceptionFilter implements ExceptionFilter {

catch(exception: BadRequestException, host: ArgumentsHost) {

const ctx = host.switchToHttp();

const response = ctx.getResponse<Response>();

const request = ctx.getRequest<Request>();

response.status(400).json({

statusCode: 400,

message: 'Validation failed',

timestamp: new Date().toISOString(),

path: request.url,

});

}

}

``



\*\*Usage\*\*:

Apply this filter to a specific methd:



```typescript

import { Post, Body, UseFilters } from '@nestjs/common';

import { ValidationExceptionFilter } from '../common/filters/method/validation-exception.filter';

@Post('create')

@UseFilters(ValidationExceptionFilter)

createUser(@Body() createUserDto: CreateUserDto) {

// Method logic

}

``



This setup ensures that any `BadRequestException` thrown within the `createUser` method is handled by the `ValidationExceptionFilte`.

---

## ✅ Benefits of Organizing Filters in `common/filters/`

- \*\*Centralized Error Handling\*: By placing all filters in a dedicated directory, you maintain a single source of truth for exception handling logc.

- \*\*Enhanced Readability\*: Categorizing filters based on their scope (global, controller,