Nestjs middleware

Big Picture (Request Lifecycle)

```
Incoming request → Guard → Interceptor (before) → Pipe → Controller →
Service →
→ Interceptor (after) → Filter (if exception) → Response
```

Each of these is a **middleware-type layer**, but they serve different purposes:

Concept	Purpose	Runs When	Example Use
Pipe	Transform or validate request data	Before controller method	Validate DTO or convert string → number
Filter	Handle exceptions and format responses	When exception is thrown	Catch and return custom error JSON
Guard	Control access (authorization)	Before route handler	Check JWT or roles
Interceptor	Wrap method execution (pre/post logic)	Around controller method	Log, cache, measure time, modify response

---

⚙️ 1️⃣ Pipe – *Validation & Transformation*

Example: `ValidationPipe` (built-in)

Registers globally:

```
app.useGlobalPipes(
  new ValidationPipe({
    whitelist: true,
    forbidNonWhitelisted: true,
    transform: true,
  }),
);
```

What it does

- Validates incoming DTOs using `class-validator`
- Removes unexpected fields (`whitelist`)
- Converts types automatically (`transform`)

Demo

Controller:

```
@Post()
create(@Body() dto: CreateEmployeeDto) {
  // If "age" is not a number, ValidationPipe throws 400
  return this.employeeService.create(dto);
```

```
}
```

DTO:

```
export class CreateEmployeeDto {
  @IsString()
  name: string;

  @IsEmail()
  email: string;

  @IsInt()
  @Type(() => Number)
  age: number;
}
```

Pipe transforms & validates `age` before controller runs.

⚙️ 2️⃣ Filter – *Error handling*

Example: Custom Exception Filter

```
import { ExceptionFilter, Catch, ArgumentsHost, HttpException } from
'@nestjs/common';
import { Response } from 'express';

@Catch(HttpException)
export class HttpErrorFilter implements ExceptionFilter {
  catch(exception: HttpException, host: ArgumentsHost) {
    const ctx = host.switchToHttp();
    const response = ctx.getResponse<Response>();
    const status = exception.getStatus();

    response.status(status).json({
      success: false,
      statusCode: status,
      message: exception.message,
      timestamp: new Date().toISOString(),
    });
  }
}
```

Apply globally:

```
app.useGlobalFilters(new HttpErrorFilter());
```

Now any thrown `HttpException` (e.g., `NotFoundException`) will be formatted neatly.

⚙️ 3️⃣ Guard – *Authorization / Access Control*

Example: Simple Auth Guard

```
import { CanActivate, ExecutionContext, Injectable } from '@nestjs/common';

@Injectable()
export class AuthGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean {
    const request = context.switchToHttp().getRequest();
    return request.headers['x-api-key'] === 'secret123'; // demo logic
  }
}
```

Apply to route:

```
@UseGuards(AuthGuard)
@Get()
findAll() {
  return this.employeeService.findAll();
}
```

If header `x-api-key` isn't correct, Nest auto-returns **403 Forbidden**.

⚙️ 4️⃣ Interceptor – *Cross-cutting behavior*

Example: Logging Interceptor

```
import {
  CallHandler, ExecutionContext, Injectable, NestInterceptor,
} from '@nestjs/common';
import { Observable, tap } from 'rxjs';

@Injectable()
export class LoggingInterceptor implements NestInterceptor {
  intercept(context: ExecutionContext, next: CallHandler): Observable<any>
{
    const req = context.switchToHttp().getRequest();
    const start = Date.now();
    console.log(`➡️ ${req.method} ${req.url}`);

    return next.handle().pipe(
      tap(() => console.log(`⬅️ Done in ${Date.now() - start}ms`)),
    );
  }
}
```

Apply globally:

```
app.useGlobalInterceptors(new LoggingInterceptor());
```

It logs before and after the controller runs.

Combined Demo (easy visualization)

When you call:

```
POST /employees
{
  "name": "Asha",
  "email": "asha@example.com",
  "role": "admin"
}
```

The flow is:

1️⃣ **Guard** checks header → rejects if no key.
2️⃣ **Interceptor** logs start time.
3️⃣ **Pipe** validates DTO (`role` will be stripped due to whitelist).
4️⃣ **Controller** executes → Service → DB.
5️⃣ **Interceptor** logs time after response.
6️⃣ If any exception → **Filter** formats the error JSON.

🧩 Summary Table

Layer	Purpose	Example
Pipe	Validate/transform input	`ValidationPipe`, `ParseIntPipe`
Filter	Handle exceptions	`HttpExceptionFilter`
Guard	Allow/deny request	`AuthGuard`, `RolesGuard`
Interceptor	Add cross-cutting behavior (before/after)	`LoggingInterceptor`, `TimeoutInterceptor`