

NestJS Assignment - 9

Theoretical Questions:

1. What is an Interceptor in NestJS?

An interceptor is a class annotated with `@Injectable()` that implements the `NestInterceptor` interface. It can bind extra logic before/after method execution like logging, transforming results, or error handling. Unlike middleware, interceptors run after route guards and before/after the method itself.

2. Explain the concept of 'method execution context' in NestJS interceptors.

The 'method execution context' includes metadata about the request, handler, class, and arguments. Interceptors access this through the `ExecutionContext` object to apply logic around the actual method call.

3. What are the main use cases for using interceptors in NestJS?

Common use cases include:

- Logging request/response
- Transforming or mapping responses
- Adding metadata
- Caching responses
- Centralized error handling

4. How can interceptors be used to handle errors in NestJS?

Interceptors can catch errors using `RxJS catchError` inside the `intercept()` method. They can modify or log the error, or throw a new custom exception.

Practical Questions:

5. Create a basic logging interceptor in NestJS.

```
PS C:\Users\Jafar\Desktop\Nodejs\js-files\day10-interceptor> nest g interceptor logging
CREATE src/logging/logging.interceptor.ts (322 bytes)
CREATE src/logging/logging.interceptor.spec.ts (203 bytes)
PS C:\Users\Jafar\Desktop\Nodejs\js-files\day10-interceptor>
```

```
src > logging > TS logging.interceptor.ts > LoggingInterceptor > intercept
1 import { CallHandler, ExecutionContext, Injectable, NestInterceptor } from '@nestjs/common';
2 import { log } from 'console';
3 import { Observable, tap } from 'rxjs';
4
5 @Injectable()
6 export class LoggingInterceptor implements NestInterceptor {
7   intercept(context: ExecutionContext, next: CallHandler): Observable<any> {
8     const request=context.switchToHttp().getRequest()
9     const url=request.url
10    const method=request.method
11    const time=new Date().toISOString()
12
13    log("Response Received!");
14
15    return next.handle().pipe(tap(()=>log(`[${time}] ${method} ${url} - Response sent`)));
16  }
17 }
18
```

GET

http://localhost:3000/

Send

Status: 200 OK

Size: 12 Bytes

Time: 25 ms

Query

Headers 2

Auth

Body

Tests

Pre Run

Response

Headers 6

Cookies

Result

1 Hello World!

Query Parameters

parameter

value

PROBLEMS

OUTPUT

TERMINAL

PORTS

DEBUG CONSOLE

[8:08:32 pm] File change detected. Starting incremental compilation...

[8:08:32 pm] Found 0 errors. Watching for file changes.

[Nest] 2156 - 14/04/2025, 8:08:33 pm LOG [NestFactory] Starting Nest application...

[Nest] 2156 - 14/04/2025, 8:08:33 pm LOG [InstanceLoader] AppModule dependencies initialized +15ms

[Nest] 2156 - 14/04/2025, 8:08:33 pm LOG [RoutesResolver] AppController {/}: +6ms

[Nest] 2156 - 14/04/2025, 8:08:33 pm LOG [RouterExplorer] Mapped {/, GET} route +5ms

[Nest] 2156 - 14/04/2025, 8:08:33 pm LOG [NestApplication] Nest application successfully started +3ms

Response Received!

[2025-04-14T14:38:34.414Z] GET / - Response sent

6. How would you use an interceptor to transform the response before sending it to the client?

- Implement a custom interceptor that converts all numeric response data into strings before sending it back to the client.

(data Passed in request is already in string format, so converting the string to numbers before sending to the client)

```
src > string-to-numeric > TS string-to-numeric.interceptor.ts > StringToNumericInterceptor > Intercept
1 import { CallHandler, ExecutionContext, Injectable, NestInterceptor } from '@nestjs/common';
2 import { Observable } from 'rxjs';
3 import { map } from 'rxjs/operators';
4
5 @Injectable()
6 export class StringToNumericInterceptor implements NestInterceptor {
7   intercept(context: ExecutionContext, next: CallHandler): Observable<any> {
8     return next.handle().pipe(map(data => {
9       data.numeric_values = JSON.parse(data.numeric_values);
10      return data
11    }));
12  }
13 }
14
15
16
```

```
16 @Post("string-to-numeric")
17 @UseInterceptors(NumericToStringInterceptor)
18 convert(@Body() body: { numeric_values: string }){
19   console.log(body.numeric_values);
20   return { numeric_values: body.numeric_values };
21 }
```

Method	URL	Status	Created	Size	Time
POST	http://localhost:3000/string-to-numeric	201	Created	36 Bytes	40 ms

Query	Headers	Auth	Body	Tests	Pre Run	
JSON	XML	Text	Form	Form-encode	GraphQL	Binary

JSON Content
Format

```

1 {
2   "numeric_values": "[1,2,3,4,5,6,7,8]"
3 }

```

Response

Headers

Cookies

Results

Docs

```

1 {
2   "numeric_values": [
3     1,
4     2,
5     3,
6     4,
7     5,
8     6,
9     7,
10    8
11  ]
12 }

```

7. Apply interceptors to specific routes or controllers.

In This we can see we can either use it in the top level i.e class level which will intercept every request, or else just use it above every controller route (Interceptor will only monitor the specific route on which it was attached)

```
1 import { Body, Controller, Get, Post, UseInterceptors } from '@nestjs/common';
2 import { AppService } from './app.service';
3 import { LoggingInterceptor } from './logging/logging.interceptor';
4 import { StringToNumericInterceptor } from './string-to-numeric/string-to-numeric.interceptor';
5
6 @Controller()
7 @UseInterceptors(LoggingInterceptor)
8 export class AppController {
9   constructor(private readonly appService: AppService) {}
10
11   @Get()
12   @UseInterceptors(LoggingInterceptor)
13   getHello(): string {
14     return this.appService.getHello();
15   }
16 }
```