Jafar Beldar

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 > logging.interceptor.ts > LoggingInterceptor > intercept

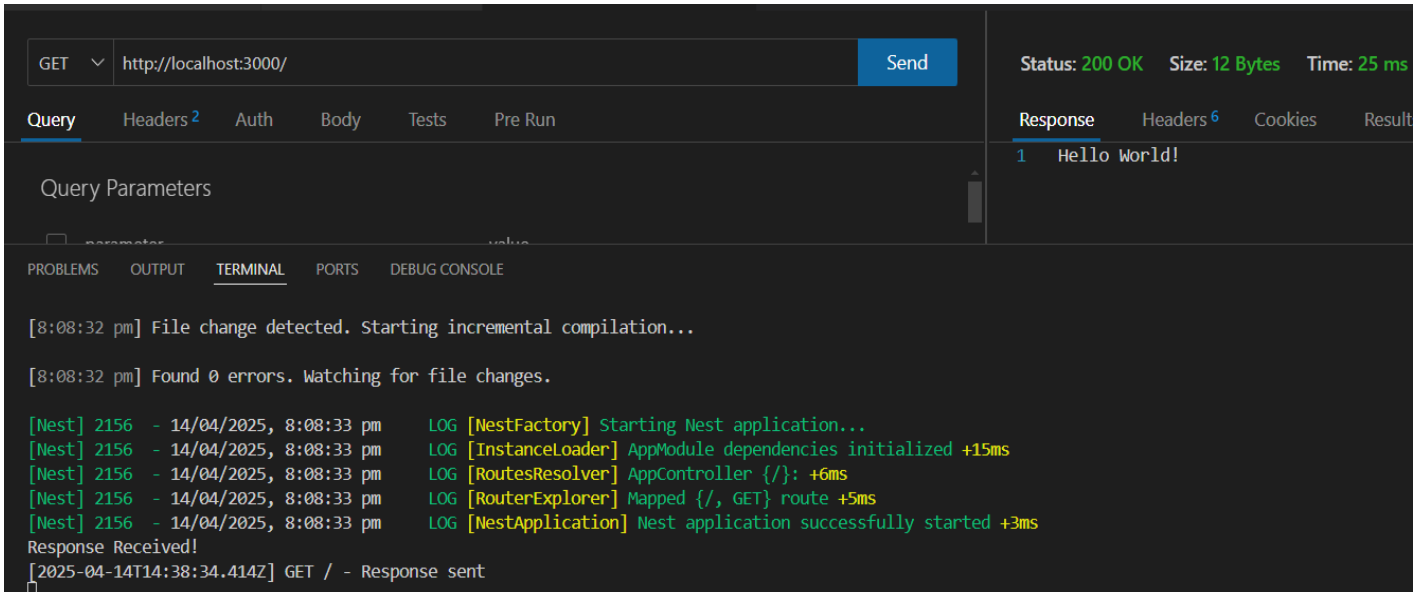
```ts
import { CallHandler, ExecutionContext, Injectable, NestInterceptor } from '@nestjs/common';
import { log } from 'console';
import { Observable, tap } from 'rxjs';

@Injectable()
export class LoggingInterceptor implements NestInterceptor {
  intercept(context: ExecutionContext, next: CallHandler): Observable<any> {
const request=context.switchToHttp().getRequest()
const url=request.url
const method=request.method
const time=new Date().toISOString()

log("Response Received!");

    return next.handle().pipe(tap(()=>log(`[${time}] ${method} ${url} - 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)

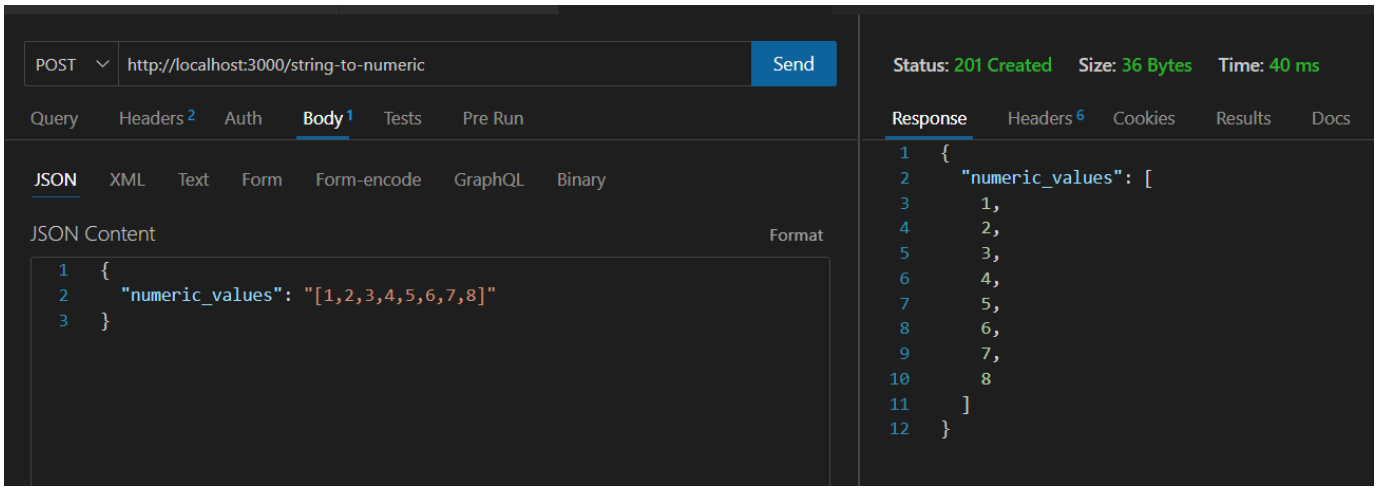
src > string-to-numeric > string-to-numeric.interceptor.ts > StringToNumericInterceptor > intercept

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

@Injectable()
export class StringToNumericInterceptor implements NestInterceptor {
    intercept(context: ExecutionContext, next: CallHandler): Observable<any> {
      return next.handle().pipe(map(data => {
                data.numeric_values = JSON.parse(data.numeric_values);
                return data
              }
            )
        );
      }
}

```

```ts
@Post("string-to-numeric")
@UseInterceptors(NumericToStringInterceptor)
convert(@Body() body: { numeric_values: string }){
  console.log(body.numeric_values);
  return { numeric_values: body.numeric_values };
}
```



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
```