How to use the Nest.js exception filter for error handling

Unhandled exceptions can cause confusion and frustration for users. However, you can solve this problem with exception filters.

Nest.js exception filters provide a way to catch and handle exceptions globally or on a per-controller basis. They allow you to focus on error handling logic, format error responses, and provide consistent error handling across the application. Let's learn about exception filters and how to use them for proper application error handling.

Default error handling in Nest.js

By default, Nest.js has an exception class, which handles any exceptions that the application code cannot handle.

When unresolved errors occur in the application, Nest.js will detect them and return a 100 Internet Server error to the client. The JSON that Nest.js returns in this case looks like this:

```
{ "statusCode": 500, "message": "Internal server error" }
```

If the error object generated by code contains **statusCode** and **message** , Nest.js will return these values ?? instead of the default response.

To avoid the common behavior of sending a more detailed error response to the client, you need to diligently handle all possible errors in the application. You can achieve this using Nest.js's built-in or custom exception filters.

Create a custom exception filter

This example will try to create a filter that handles all HTTP exceptions. Start with a file named **http.exception.ts** and add the following imports to it:

```
import { ExceptionFilter, Catch, ArgumentsHost, HttpException, } from '@nestjs/c
```

These imports serve the following purposes:

1. **ExceptionFilter** : an interface that describes an exception filter implementation.
2. **Catch** : A decorator that marks the class as a Nest exception filter.
3. **ArgumentsHost** : This interface provides methods for retrieving arguments passed to a handler. It allows you to select the appropriate execution body (for example, HTTP, RPC, or WebSockets) to retrieve arguments.
4. **HttpException** : This is a class that defines basic HTTP Nest exceptions.
5. **Request & Response** : They are the interfaces for an Express.js query and the corresponding response object.

Next, create a class, **HttpExceptionFilter** , that implements **ExceptionFilter** . Annotate it with the **Catch** decorator to indicate that it handles HttpExceptions:

```
@Catch(HttpException) export class HttpExceptionFilter implements ExceptionFilte
```

Next, fill the class with this code:

```
catch(exception: HttpException, host: ArgumentsHost) { // L?y ??i t??ng ph?
n h?i t? máy ch? ??i s?
 const ctx = host.switchToHttp(); const response = ctx.getResponse(); // L?
y ??i t??ng ph?n h?i t? máy ch? ??i s?
 const request = ctx.getRequest(); // L?y code tr?ng thái t? ngo?i l?
 const status = exception.getStatus(); // G?i ph?n h?i JSON b?ng ??i t??
ng ph?n h?
i response.status(status).json({ statusCode: status, timestamp: new Date().toISO
```

This block of code retrieves the query and response objects from the ArgumentsHost object and retrieves related information from this exception. It returns a JSON object response, with error details, to the client.

Link exception filters

You can associate an exception filter with a driver or the entire application, as needed.

To bind a global exception filter, first import the exception filter into the **main.ts** file . Then pass the instance of the exception filter into the **app.useGlobalFilters** method :

```
// main.ts import { NestFactory } from '@nestjs/core'; import { AppModule } from
?t b? l?c v?i ?ng d?
ng app.useGlobalFilters(new HttpExceptionFilter()); await app.listen(4050); } bo
```

To bind an exception to a controller, enter the **UseFilters** decorator and your exception filter. Annotate the controller class with the **@UseFilters** decorator and pass the instance of the exception filter as an argument to the decorator:

```
@Controller() @UseFilters(new HttpExceptionFilter()) export class AppController
```

Where you bind the filter determines the scope of error handling. Controller binding filters will only serve the controller you're binding to, app binding filters will serve the entire app.

Above is **how to use Nest.js's exception filter to handle errors** . Hope the article is useful to you.