Documentation & Validation

Qinshift Academy

Validation

Validation is the process of ensuring that incoming data is correct, complete, and in the expected format before your application processes it.

Why is validation important?

- Prevents bad or malicious data from entering your system
- Helps avoid runtime errors and unexpected behavior
- Improves application reliability and security
- Makes your API self-defensive and predictable

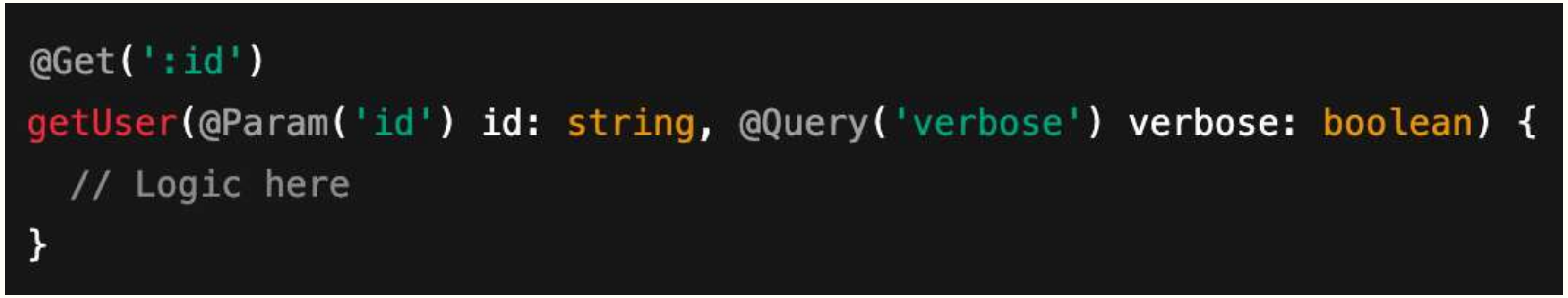
How does validation work in NestJS

- Validation is done using DTOs and the class-validator library
- Works with the ValidationPipe to automatically check request data
- If the data is invalid, NestJS will return a 400 Bad Request with helpful error messages

Handling Request Data in NestJS

- NestJS allows us to extract and work with request data using special decorators:

Decorator	Extracts data from
@Body()	Request body (POST, PUT...)
@Query()	URL query string
@Param()	Route parameters



```
@Get(':id')
getUser(@Param('id') id: string, @Query('verbose') verbose: boolean) {
  // Logic here
}
```

What is a DTO?

DTO = Data Transfer Object

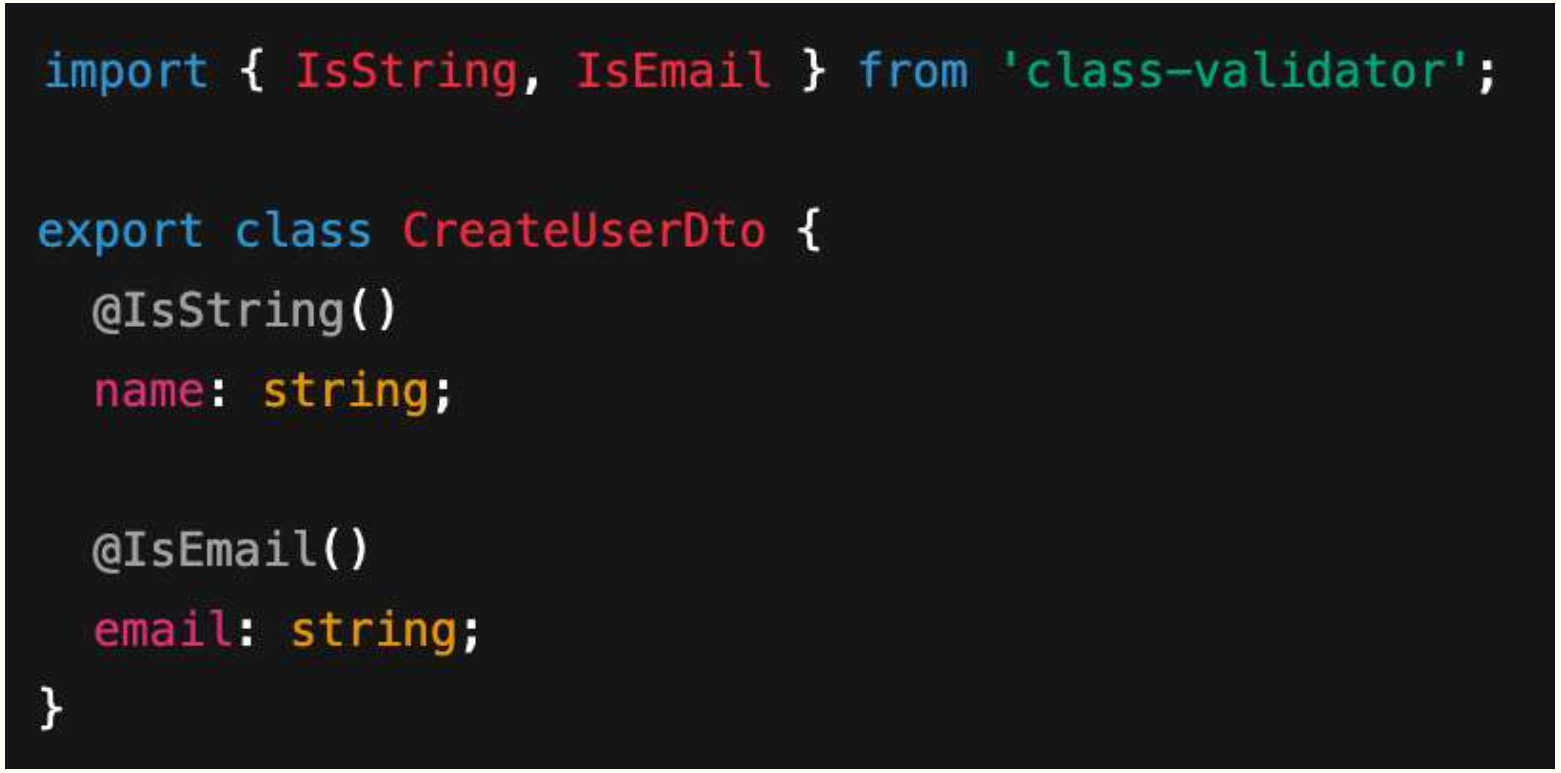
A DTO is a TypeScript class that defines the shape and type of incoming data.

Helps with:

- Validation
- Type safety
- Data transformation
- Code readability

Validating Data with DTOs

NestJS integrates with class-validator and class-transformer to validate and transform data using decorators in your DTO class.

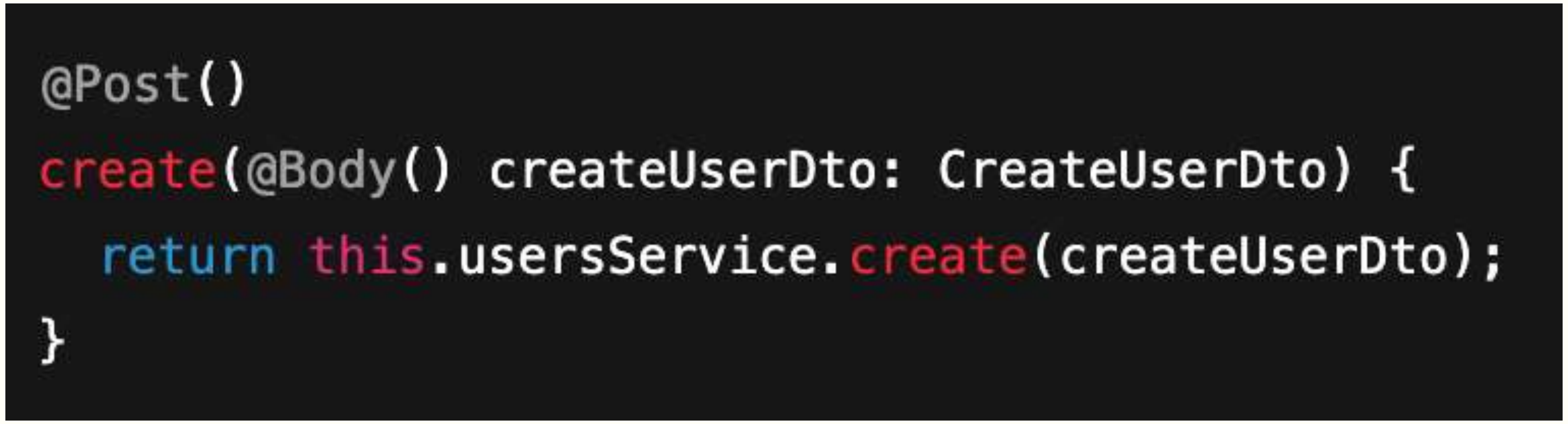


```
import { IsString, IsEmail } from 'class-validator';

export class CreateUserDto {
  @IsString()
  name: string;

  @IsEmail()
  email: string;
}
```

Applying DTOs to Endpoints

Nest will automatically:

- Validate the incoming request body
- Reject requests with invalid data (if ValidationPipe is enabled)



```
@Post()
create(@Body() createUserDto: CreateUserDto) {
  return this.usersService.create(createUserDto);
}
```

What is Data Transformation?

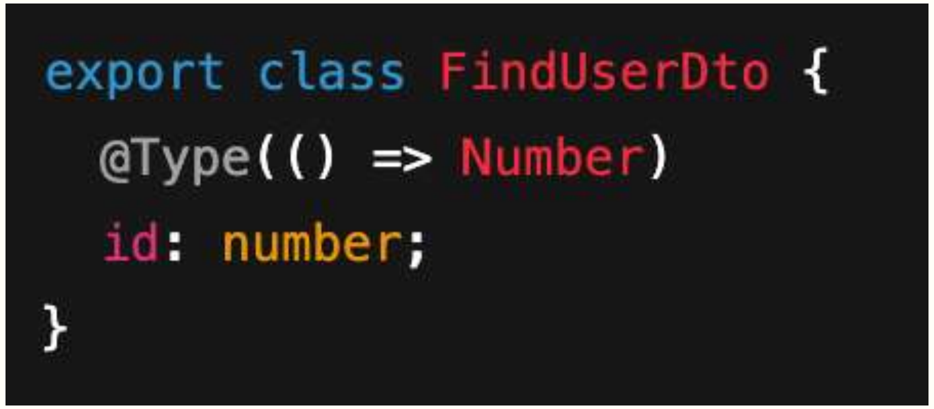
Data Transformation is the process of automatically converting incoming request data into the expected data types or class instances before it's used in your application.

Why It Matters

- Ensures you’re working with correct types (e.g., number instead of string)
- Converts raw JSON into class instances (e.g., a DTO class)
- Makes validation and business logic more reliable
- Reduces the need for manual parsing/casting

How It Works in NestJS

NestJS uses the class-transformer library in combination with the ValidationPipe to transform request data.



```
export class FindUserDto {
  @Type(() => Number)
  id: number;
}
```

If a client sends a query like ?id=5, the string "5" will automatically be transformed into the number 5.

Documentation

Documentation is a way to describe your API so developers can understand how to use it — what endpoints exist, what data they expect, and what they return.

What is Swagger?

Swagger (now part of the OpenAPI standard) is a toolset and specification for describing RESTful APIs.

It provides an interactive UI where users can:

- See available routes
- Understand input/output formats
- Test endpoints directly

NestJS + Swagger

NestJS offers built-in support for Swagger via the @nestjs/swagger package.

- ✓ Auto-generates docs from your decorators
- ✓ Reflects your DTOs, types, and routes
- ✓ Useful for frontend devs, testers, and clients

Questions?

Trainer Name

Trainer

trainer@mail.com

Assistant Name

Assistant

asistant@mail.com

Qinshift Academy