

# Validation

The validation is an essential functionality of any existing web application. To automatically validate incoming requests, we leverage **class-validator** package which built-in `ValidationPipe` makes use of underneath. The `ValidationPipe` provides a convenient method to validate incoming client payloads with a variety of powerful validation rules.

## Overview

In **Pipes** chapter, we went through the process of building a simplified validation pipe. To better understand what we're doing under the hood, we heavily recommend to read this article. Here, we'll mainly focus on the real use-cases instead.

## Auto-validation

For the sake of this tutorial, we'll bind `ValidationPipe` to the whole application, thus, all endpoints will be automatically protected from the incorrect data.

```
async function bootstrap() {
  const app = await NestFactory.create(ApplicationModule);
  app.useGlobalPipes(new ValidationPipe());
  await app.listen(3000);
}
bootstrap();
```

To test our pipe, let's create a basic endpoint.

```
@Post()
create(@Body() createUserDto: CreateUserDto) {
  return 'This action adds a new user';
}
```

Then, add a few validation rules in our `CreateUserDto`.

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

export class CreateUserDto {
  @IsEmail()
  email: string;
```

```
@IsNotEmpty()  
password: string;  
}
```

Now, when someone requests our endpoint with an invalid `email`, the application would respond with a `400 Bad Request` code and the following response body.

```
{  
  "statusCode": 400,  
  "error": "Bad Request",  
  "message": [  
    {  
      "target": {},  
      "property": "email",  
      "children": [],  
      "constraints": {  
        "isEmail": "email must be an email"  
      }  
    }  
  ]  
}
```

Obviously, the response body is not a sole use-case for the `ValidationPipe`. Imagine, that we would like to accept `:id` in the endpoint path. Only numbers shall be valid though. This is pretty simple as well.

```
@Get('/:id')  
findOne(@Param() params: FindOneParams) {  
  return 'This action returns a user';  
}
```

And `FindOneParams` looks like below.

```
import { IsNumberString } from 'class-validator';  
  
export class FindOneParams {  
  @IsNumberString()  
  id: number;  
}
```

## Disable detailed errors

Error messages help a lot in order to comprehend what was wrong with the data sent through the network. However, in some production environments, you might want to disable detailed errors.

```
app.useGlobalPipes(  
  new ValidationPipe({  
    disableErrorMessage: true,  
  }),  
);
```

Now, error messages won't be populated to the end user.

## Stripping properties

Quite frequently, we would like only predefined (whitelisted) properties to be passed on. For instance, if we expect `email` and `password` properties, when someone sends `age`, this property should be stripped and not available in the DTO. To enable such behavior, set `whitelist` to `true`.

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

This setting will enable auto-stripping of non-whitelisted (without any decorator on top of them) properties. However, if you want to stop the request processing entirely, and return an error response to the user, use `forbidNonWhitelisted` instead.

## Auto payload transforming

The `ValidationPipe` doesn't automatically transform your payloads to the corresponding DTO classes. If you take a look at either `createUserDto` or `findOneParams` in your controller methods, you will notice that they're not actual instances of these classes. To enable auto-transformation, set `transform` to `true`.

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

## Websockets & Microservices

All these guidelines concern both WebSockets as well as microservices, regardless of transport method that is being used.

### Learn more

In order to read more about custom validators, error messages, and available decorators, visit this [page](#).

---

## Support us

Nest is an MIT-licensed open source project. It can grow thanks to the support by these awesome people. If you'd like to join them, please read more [here](#).

### Principal Sponsor



### Sponsors / Partners

