ValidationPipeOptions

August 18, 2026

ValidationPipeOptions

Kind: Interface

Source: `packages/common/pipes/validation.pipe.ts`

Part of: Common

`ValidationPipeOptions` configures how NestJS's `ValidationPipe` validates incoming request data and transforms payloads into typed DTO instances. It controls validation behavior, error responses, class-transformer integration, and optional custom validator or transformer package implementations.

Properties

Property	Type
`transform`	`boolean`
`disableErrorMessages`	`boolean`
`transformOptions`	`ClassTransformOptions`
`errorHttpStatusCode`	`ErrorHttpStatusCode`
`exceptionFactory`	`(errors: ValidationError[]) => any`
`validateCustomDecorators`	`boolean`
`expectedType`	`Type<any>`
`validatorPackage`	`ValidatorPackage`
`transformerPackage`	`TransformerPackage`

Diagram

```
graph LR
  Request[Incoming request payload] --> Pipe[ValidationPipe]
```

```
  Pipe --> Transform{transform enabled?}
  Transform -- Yes --> DTO[Transform payload to DTO instance]
  Transform -- No --> Validate[Validate raw payload]
  DTO --> Validate
  Validate --> Valid{Validation succeeds?}
  Valid -- Yes --> Handler[Route handler]
  Valid -- No --> Factory[exceptionFactory]
  Factory --> Error[HTTP error response]

  Options[ValidationPipeOptions] --> Pipe
  Options --> Transform
  Options --> Validate
  Options --> Factory
```

Usage

```
import {
  BadRequestException,
  ValidationPipe,
  ValidationError,
} from '@nestjs/common';
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(
    new ValidationPipe({
      transform: true,
      transformOptions: {
        enableImplicitConversion: true,
      },
      disableErrorMessages: false,
      validateCustomDecorators: true,
      exceptionFactory: (errors: ValidationError[]) => {
        return new BadRequestException({
          message: 'Request validation failed',
          errors,
        });
      },
    }),
  );

  await app.listen(3000);
}

bootstrap();
```

AI Coding Instructions

- Enable `transform` when controllers expect DTO instances or typed primitive conversion, and configure `transformOptions` for behaviors such as implicit conversion.
- Use DTO validation decorators from `class-validator`; set `validateCustomDecorators` when custom decorators must be evaluated by the pipe.
- Provide an `exceptionFactory` when the application requires a consistent validation-error response format.
- Avoid enabling detailed error messages in production when validation errors could expose internal request or DTO details; use `disableErrorMessages` where appropriate.
- Use `validatorPackage` and `transformerPackage` only when integrating compatible custom implementations of `class-validator` or `class-transformer`.