Response

August 17, 2026

Response

Kind: Function

Source: `packages/common/decorators/http/route-params.decorator.ts`

Part of: Common

Route handler parameter decorator. Extracts the `Response` object from the underlying platform and populates the decorated parameter with the value of `Response`.

Example: `logout(@Response() res)`

`Response` is a route handler parameter decorator that injects the underlying platform response object into a controller method parameter. Use it when a handler needs direct access to response APIs, such as setting headers, cookies, status codes, or manually sending a response.

Signature

```
function Response(options: ResponseDecoratorOptions)
```

Parameters

Name	Type
`options`	`ResponseDecoratorOptions`

Diagram

```
graph LR
  A[Incoming HTTP Request] --> B[Route Handler]
  B --> C[@Response() Decorator]
  C --> D[Platform Response Object]
```

```
  D --> E[Decorated Handler Parameter]
  E --> F[Set headers, status, cookies, or send response]
```

Usage

```ts
import { Controller, Get, Response } from '@nestjs/common';

@Controller('auth')
export class AuthController {
  @Get('logout')
  logout(@Response() res: any) {
    res.clearCookie('session');
    return res.status(200).send({ message: 'Logged out successfully' });
  }
}
```

AI Coding Instructions

- Use `@Response()` only when the route handler requires direct access to the platform-specific response object.
- When manually sending a response with methods such as `res.send()` or `res.json()`, ensure the handler does not also return a framework-managed response.
- Keep response-specific logic limited to controllers; move business logic into services where possible.
- Account for adapter differences when supporting multiple HTTP platforms, such as Express and Fastify.
- Prefer standard return values for simple handlers to preserve framework-managed serialization and status handling.