Controllers

Controllers in nestjs are responsible for handling incoming http requests and returning responses to the client. A controller main purpose is to receive specific requests for the application. The routing mechanism controls which controller receives which request. Frequently, each controller has more than one route, and different routes can perform different actions.

In order to create a basic controller, we use classes and decorators. Decorators associate classes with required metadata and enable nest to create a routing map.

Characteristics

* Decorated with @Controller() decorator.
* Define routes using http method like GET,POST,PUT,PATCH, DELETE and oath parameters.
* Typically interact with services to perform business logic
* Return responses to the client

Providers

Providers are a fundamental concept in Nest. Many of the basic nest classes may be treated as a provider like services, repositories, factories, helpers. The main idea of a provider is that it can be injected as a dependency.

Characteristics

* Can be injected into other providers / controller
* Managed by nest dependency injection system

Benefits of using controllers and providers

* Improved code organization
* Enhanced testability
* Increased reusability
* Better dependency management

Modules

Modules are the fundamental building block for organizing our nestjs application. They provide a way to group related component like controller, providers, and other modules into cohesive units. This promotes code reusability, maintainability, and testability.

Characteristics of modules

Encapsulation

Dependency injection

Reusability

Organization

Structure of module

A module is a class decorated with the @Module () decorator. The decorator provides metadata about the module, such as its imports, exports, controllers, and providers.

imports

array of modules that this module depends on.

exports

array of providers / modules that can be used by other modules.

controllers

array of controllers defined in this module

providers

array of providers defined in this module

modules relationship

import

a module can import another module to access its exported provider / module

export

a module can export its providers / modules to be used by other modules.

Shared providers

Providers are singletons by default, so they are shared across all modules that import them.

Type of modules

Regular modules

The most common type, used for organizing application logic.

Dynamic module

More complex, used for scenario like custom module loaders / asynchronous module creation.

Best practices

Small, focused module with clear responsibilities

Shared modules for common components use across multiple modules

Lazy loading large module to improve performance

Middleware

Exception filters

Pipes

Guards

Interceptors

Custom decorators

Modifying response

Modifying request and response in Nestjs

Route handlers

Directly manipulate the response object within the route handler

Set headers, status codes and body content as needed.

Custom middleware

Modify the response object within middleware function

Useful for global response modification / specific logic.

Transform Interceptor

Specifically designed for transforming response data.

Provides a clean separation of concerns

Modifying request

Custom middleware

Modify request objects before they reach the route handler

Useful for adding / removing request headers, body parameter.

Pipe

Transform request data, specifically focusing on input validation and transformation

Transform interceptor

Modify request data, but is typically used for response transformation.

Interceptor are for global modification

Route handler for route specific modification

Pipe for data transformation

Middleware for general request / response