Nestjs

A progressive nodejs framework for building efficient, reliable and scalable server-side applications.

Setting up nest cli

**>npm I –g @nestjs/cli**

Setting up new nestjs

**>nest new project-name**

**Folder structure**

node\_modules

src

app.controller.spec.ts

app.controller.ts

app.module.ts

app.service.ts

main.ts

test

.eslintrc.js

Nestjs modules

A module is a class annotated with a @Module () decorator. The @Module () decorator provides metadata that **nest** makes use of to organize the application structure.

orders module

chat module

feature module 3

feature module 2

feature module 1

user module

Application module

Nestjs controller

Controller are responsible for handling incoming request and returning responses to the client.

controller

controller

client side

http request

controller

Nestjs provider

Providers are a fundamental concept in nest. Many of the basic nest classes may be treated as provider services, repositories, factories, helpers and so on.

The main idea of provider is that it can be injected as dependency.

Nestjs pipes

A pipe is a class annotated with the @Injectable () decorator, which implements the PipeTransform Interface.

Pipe have two use case  
1. Transformation

Transform input data to the desired form.

2. validation

evaluate input data and if valid, simply pass it through unchanged, throw an

exception when the data is incorrect.

Nestjs guard

A guard is a class annotated with the @Injectable () decorator, which implements the CanActivate interface.

Guards have a single responsibility. They determine whether a given request will handle by the route handler or not, depending on certain condition (like permissions, roles, ACLs, etc.) present at runtime.

Route Handler

Guard

http request

Client side

Nestjs interceptor

A interceptor is a class annotated with the @Injectable () decorator, which implements the NestInterceptor interface.

request

Client side

@Get()

Route handler

response

Interceptors have a set of useful capabilities which are inspired by the Aspect oriented programming (AOP) technique.

Bind extra logic before/after method execution And transform the result returned from a function And transform the exception thrown from a function And extend the basic function behavior And completely override a function depending on specific conditions (e.g. for caching purpose)