Some questions to prepare you for

NestJs Interview

dalanda-drissi

What is NestJS and why choose it over other Node.js frameworks?

NestJS is a TypeScript-based, modular Node.js framework. It offers features like **dependency injection**, **modular architecture**, and **strong community support**, making it ideal for **building scalable and maintainable applications.**

What is a module in NestJS?

A module is a class annotated with **@Module**() that organizes related components (controllers, providers, etc.) into a cohesive block, improving code modularity and reusability.

How does NestJS handle Dependency Injection?

NestJS uses **@Injectable**() to register classes in the DI system. Dependencies are injected via the constructor, promoting loose coupling and easier testing.

What are Guards in NestJS?

Guards control whether a request is processed, often used for authentication and authorization. Implement the CanActivate interface to define custom logic.

What are Interceptors in NestJS?

Interceptors manipulate request/response data, useful for logging or transforming responses. They are applied globally or at the controller/method level.

What are Pipes in NestJS and how are they used?

Pipes are used to transform or validate data before it reaches the route handler. They can be applied globally, at the controller, or at the method level.

example : *@UsePipes(new ValidationPipe())*

What are Providers in NestJS?

Providers are classes annotated with **@Injectable**() and are responsible for handling business logic. They can be injected into other components using NestJS's DI system.

How does Exception Handling work in NestJS?

NestJS uses Exception Filters to catch and handle exceptions globally or locally. HttpException is a common built-in exception.

example : `throw new HttpException('Forbidden', 403);`

What are Middlewares in NestJS?

Middlewares are functions that run before route handlers. They can be used for tasks like logging or request modification.

example : `app.use(loggerMiddleware);`

How does NestJS implement testing?

NestJS provides utilities for unit and end-to-end testing via Jest. It supports DI for mocking dependencies and comes with @nestjs/testing for creating test modules.

What are Decorators in NestJS?

Decorators are metadata annotations, used to modify the behavior of classes, methods, and parameters. Examples include **@Controller**(), **@Get**(), **and @Injectable**().

What is the purpose of @Param() in NestJS?

@Param() is a decorator that extracts route parameters from the URL in controllers. It can be used to access parameters passed in the URL dynamically.

```
@Get(':id') findById(@Param('id') id: string)
{ return this.service.find(id); }
```

What is the role of Controllers in NestJS?

Controllers handle incoming HTTP requests and send responses back to the client. They are decorated with **@Controller**() and use methods like @Get(), @Post().

How does NestJS support Microservices?

NestJS supports microservices via built-in transport layers (e.g., Redis, gRPC). It uses ClientProxy to handle communication between services.

What are the lifecycle events in NestJS?

NestJS has lifecycle hooks like **onModuleInit** and **onApplicationBootstrap** to perform actions when modules are initialized or the application is bootstrapped.

What is the ConfigModule in NestJS?

The ConfigModule is used for managing environment variables and configuration. It loads .env files and provides values throughout the application using **@Inject(ConfigService).**

What is the Request Lifecycle in NestJS?

In NestJS, the request lifecycle includes Middleware, Guards, Interceptors, Pipes, and Exception Filters, which process the request sequentially before reaching the controller and its handlers.

What is the @Injectable() decorator used for?

@Injectable() marks a class as a provider, which can be injected into other classes using NestJS's DI system. This helps manage dependencies across modules.

What are Dynamic Modules in NestJS?

Dynamic modules allow modules to be configured asynchronously or provide different configurations at runtime. They return a module from a static method, **forRoot() or forRootAsync().**

How does NestJS handle asynchronous programming?

NestJS uses **async/await and Promises** to handle asynchronous operations, making it compatible with modern async workflows, such as working with databases or external APIs.

What is @Body() used for in NestJS?

@Body() extracts the JSON body of a request and passes it to the route handler. It’s commonly used in POST and PUT requests.

How do you handle file uploads in NestJS?

NestJS uses the @nestjs/platform-express package along with Multer to handle file uploads via the FileInterceptor.

```
@Post('upload')
@UseInterceptors(FileInterceptor('file'))
uploadFile(@UploadedFile() file:
Express.Multer.File) { return file; }
```

What is @UseInterceptors() in NestJS?

@UseInterceptors() applies interceptors to a method or controller. Interceptors can modify request/response data or handle logging, caching, and more.

What is the difference between @Get() and @Post() in NestJS?

- **@Get**() handles HTTP GET requests, typically used for fetching data.
- **@Post**() handles HTTP POST requests, used for creating resources or submitting data.

What is the role of @Global() in NestJS?

@Global() makes a module globally available across the entire application, so it doesn’t need to be imported in every module that uses its providers.

dalanda-drissi

How do you enable Cross-Origin Resource Sharing (CORS) in NestJS?

CORS can be enabled using the **app.enableCors**() method in the main bootstrap() function or by configuring it in the NestFactory options.