1. Module trong NestJS là gì? Tại sao nó quan trọng?

Module trong NestJS là một đơn vị tổ chức mã nguồn, giúp nhóm các tính năng liên quan lại với nhau. Mỗi ứng dụng trong NestJS đều có ít nhất một module (thường là `AppModule`), và bạn có thể tạo thêm nhiều module để chia nhỏ ứng dụng theo các chức năng riêng biệt (ví dụ: `UsersModule`, `AuthModule`, `ProductsModule`, ...). Module giúp dễ dàng quản lý và tổ chức mã nguồn, đồng thời giúp cho việc tái sử dụng, kiểm tra và bảo trì mã nguồn trở nên hiệu quả hơn.

2. Sự khác biệt giữa `@Module()`, `@Global()`, và `@Injectable()`?

- **`@Module()`**: Là một decorator dùng để khai báo các module trong NestJS. Bạn sử dụng `@Module()` để định nghĩa các providers, controllers, và các module phụ thuộc.
- `@Module({`
- `  controllers: [AppController],`
- `  providers: [AppService],`
- `})`
- `export class AppModule {}`
- **`@Global()`**: Dùng để đánh dấu module là "global", có nghĩa là các providers của module đó sẽ có sẵn trong toàn bộ ứng dụng mà không cần phải import vào các module khác.
- `@Global()`
- `@Module({`
- `  providers: [AppService],`
- `})`
- `export class GlobalModule {}`
- **`@Injectable()`**: Là decorator dùng để đánh dấu một lớp là "provider", cho phép NestJS quản lý lifecycle của nó và cung cấp dependency injection (DI). Các service, repository, hoặc bất kỳ provider nào cần DI đều sẽ được đánh dấu bằng `@Injectable()`.
- `@Injectable()`
- `export class AppService {`
- `  // logic của service`
- `}`

3. Làm thế nào để import và export các providers giữa các module?

Để chia sẻ các providers giữa các module, bạn cần sử dụng `exports` trong module và `imports` trong module khác:

- **Export** các providers trong module:
- `@Module({`
- `  providers: [AppService],`
- `  exports: [AppService], // Export để các module khác có thể sử dụng`
- `})`
- `export class AppModule {}`
- **Import** các module trong module khác:
- `@Module({`
- `  imports: [AppModule], // Import AppModule để sử dụng các providers đã export`

- `})`
- `export class AnotherModule {}`

4. Controller trong NestJS đóng vai trò gì?

Controller trong NestJS là nơi xử lý các yêu cầu HTTP đến từ client. Nó nhận các yêu cầu (HTTP requests), gọi các service để xử lý logic, và trả về kết quả (HTTP response). Mỗi controller được gắn với một route cụ thể và các phương thức HTTP (GET, POST, PUT, DELETE).

5. Cách sử dụng các decorator như `@Get()`, `@Post()`, `@Param()`, và `@Body()`?

- **`@Get()`, `@Post()`**: Các decorator này dùng để xác định các phương thức HTTP. `@Get()` dành cho các yêu cầu GET, `@Post()` dành cho các yêu cầu POST.
- **`@Param()`**: Dùng để lấy các tham số từ URL của yêu cầu.
- **`@Body()`**: Dùng để lấy dữ liệu từ body của yêu cầu POST.

Ví dụ:

```
@Controller('users')
export class UsersController {
  @Get()
  findAll() {
    return 'This will return all users';
  }

  @Post()
  create(@Body() createUserDto: CreateUserDto) {
    return 'This will create a new user';
  }

  @Get(':id')
  findOne(@Param('id') id: string) {
    return `This will return user with id ${id}`;
  }
}
```

6. Provider trong NestJS là gì? Có những loại provider nào?

Provider là những đối tượng mà NestJS sẽ quản lý và cung cấp thông qua Dependency Injection (DI). Các provider có thể là các service, repository, factory, hoặc các đối tượng bất kỳ được định nghĩa trong ứng dụng.

- **Types of Providers**:
  - **Service**: Các class dùng để chứa các logic nghiệp vụ.
  - **Factory**: Cung cấp một cách tiếp cận linh hoạt để tạo ra một provider.
  - **Value**: Cung cấp một giá trị cố định.
  - **Class**: Một lớp có thể được cung cấp và được quản lý bởi NestJS.

7. Sự khác biệt giữa `@Injectable()` và `@Inject()`?

- **`@Injectable()`**: Được sử dụng để đánh dấu một lớp là "provider", cho phép nó được DI vào các class khác.
- **`@Inject()`**: Dùng trong các constructor của class để yêu cầu một phụ thuộc (dependency) cụ thể nếu bạn không thể để NestJS tự động xác định. Điều này hữu ích khi bạn cần cung cấp một tên hoặc một token đặc biệt cho dependency.

```
@Injectable()
export class MyService {
  constructor(@Inject('CustomToken') private customService: CustomService) {}
}
```

8. Middleware là gì?

Middleware là các hàm xử lý logic trước khi yêu cầu HTTP được truyền đến các controller. Middleware có thể thực hiện các tác vụ như xác thực, ghi log, hoặc xử lý lỗi trước khi yêu cầu được xử lý bởi controller.

9. Làm thế nào để tạo một custom middleware trong NestJS?

Để tạo một custom middleware, bạn cần tạo một class và implement interface `NestMiddleware`.

```
import { Injectable, NestMiddleware } from '@nestjs/common';
import { Request, Response, NextFunction } from 'express';

@Injectable()
export class LoggerMiddleware implements NestMiddleware {
  use(req: Request, res: Response, next: NextFunction) {
    console.log('Request...'); // Log request
    next();
  }
}
```

10. NestJS cung cấp những built-in middleware nào?

- **LoggerMiddleware**: Ghi log các yêu cầu HTTP.
- **BodyParserMiddleware**: Phân tích body của yêu cầu.
- **CorsMiddleware**: Hỗ trợ CORS (Cross-Origin Resource Sharing).

11. Cách sử dụng middleware trong NestJS?

Middleware có thể được áp dụng toàn cục hoặc chỉ áp dụng cho một route cụ thể.

- **Toàn cục**: Được cấu hình trong `main.ts`:
- `app.use(LoggerMiddleware);`
- **Cục bộ**: Được cấu hình trong module:
- `@Module({`

```
    providers: [LoggerMiddleware],
  })
  export class AppModule implements NestModule {
    configure(consumer: MiddlewareConsumer) {
      consumer.apply(LoggerMiddleware).forRoutes('users');
    }
  }
```

12. Dependency Injection (DI) là gì?

Dependency Injection (DI) là một kỹ thuật trong đó các đối tượng (dependencies) được "tiêm" vào các lớp (classes) thay vì tự tạo ra chúng. Điều này giúp quản lý các phụ thuộc dễ dàng hơn và giảm sự phụ thuộc lẫn nhau giữa các class.

13. Lifecycle Hooks trong NestJS là gì?

Lifecycle Hooks là các phương thức đặc biệt mà bạn có thể triển khai trong các class provider để điều khiển hành vi của chúng trong suốt vòng đời của chúng. Một số lifecycle hooks phổ biến:

- **`onModuleInit()`**: Được gọi khi module được khởi tạo.
- **`onModuleDestroy()`**: Được gọi khi module bị hủy.
- **`beforeApplicationShutdown()`**: Được gọi trước khi ứng dụng tắt.

14. ConfigModule là gì?

ConfigModule là một module trong NestJS cho phép bạn dễ dàng quản lý các cấu hình của ứng dụng, ví dụ như các biến môi trường (environment variables). Nó giúp quản lý cấu hình ứng dụng theo cách dễ dàng và linh hoạt.

```
@Module({
  imports: [ConfigModule.forRoot()],
})
export class AppModule {}
```