NestJS - Practice Sheet

10 Q&A on modules, DI, guards, pipes, interceptors, and testing

Q1. How does NestJS's modular architecture work?

A: A Module groups related controllers and providers. @Module({imports, controllers, providers, exports}) defines its boundaries. The AppModule is the root. Feature modules import other modules and export providers they want to share with importers.

Q2. How does NestJS's dependency injection container work?

A: Providers declared in a module's providers array are added to the DI container. NestJS resolves constructor injection by type token. Provider scope can be DEFAULT (singleton per module), REQUEST (new instance per request), or TRANSIENT (new per injection).

Q3. What are NestJS decorators and how do they work?

A: Decorators are TypeScript metadata annotations. @Get('/path') stores route metadata on the method. @Body(), @Param(), @Query() bind request data. NestJS's RouterExplorer reads this metadata at startup to register Express/Fastify routes automatically.

Q4. How do NestJS Pipes work and how do you validate request bodies?

A: Pipes transform or validate data before it reaches the handler. Global ValidationPipe with whitelist:true strips undeclared properties and transforms payloads. class-validator decorators (@IsEmail(), @MinLength(8)) on DTO classes define validation rules.

Q5. What are NestJS Guards and how do you implement RBAC?

A: A Guard implements CanActivate. Return true to allow or false to deny. @UseGuards(RolesGuard) applies it to a controller or method. The guard reads @Roles('admin') metadata via Reflector and checks the authenticated user's roles against the required roles.

Q6. What is a NestJS exception filter and how does it standardize error responses?

A: Implement ExceptionFilter, annotate with @Catch(MyException), and define catch(exception, host). It intercepts thrown exceptions and returns a structured JSON error response. Register globally with app.useGlobalFilters(new MyFilter()) for consistent API error shapes.

Q7. What are NestJS interceptors used for?

A: Interceptors wrap request handling before and after. The intercept(ctx, next) method calls next.handle() which returns an Observable. Use RxJS operators to transform the response (map), measure execution time, add caching, or log requests and responses.

Q8. How do you integrate TypeORM with NestJS?

A: Import TypeOrmModule.forRoot(options) in AppModule. Import TypeOrmModule.forFeature([UserEntity]) in the feature module to get a repository. Inject InjectRepository(UserEntity) private userRepo: Repository<UserEntity> into services.

Q9. How does NestJS support microservice communication?

A: NestJS microservices use ClientProxy to send messages and emit events. Transport strategies include TCP, Redis, NATS, Kafka, and RabbitMQ. @MessagePattern('pattern') on a handler receives messages. Both request-response and event-driven patterns are supported.

Q10. How do you write unit tests in NestJS?

A: Use Test.createTestingModule({providers: [MyService, {provide: Repo, useValue: mockRepo}]}).compile(). This creates a mini DI container. Call module.get(MyService) to retrieve the instance. Jest spies and mocks replace real dependencies.