

**TESTE TÉCNICO DESENVOLVEDOR DE SOFTWARE**

**Estrutura**

Parte 1: Questões teóricas (responder por escrito)

Parte 2: Desafio prático (entrega de código em repositório público ou compactado)

### **PARTE 1 — QUESTÕES TEÓRICAS (Dissertativas)**

**Objetivo:** Avaliar o raciocínio técnico, clareza de explicação e domínio conceitual.

**1. Padrões e Boas Práticas**

a) Explique o princípio de **Inversão de Dependência (D)** dentro do SOLID e descreva um exemplo prático de como ele pode ser aplicado em um projeto Node.js.

b) Cite e explique brevemente **três Design Patterns** que você já utilizou e o contexto em que eles foram aplicados.

**2. Arquitetura de Software**

a) Descreva com suas palavras as diferenças entre **REST, SOA e EDA**, e em quais cenários cada uma se encaixa melhor.

b) Como o **DDD (Domain-Driven Design)** auxilia na criação de sistemas escaláveis e alinhados ao negócio? Cite um exemplo prático de uso de “Aggregates” e “Entities”.

**3. Banco de Dados**

a) Quais critérios você utiliza para escolher entre um banco **relacional** e um **não relacional**?

b) Dê um exemplo de modelagem simples em ambos os casos (SQL e NoSQL).

**4. Testes de Software**

a) Explique o que é **TDD (Test-Driven Development)** e descreva brevemente o ciclo de desenvolvimento usando essa prática.

b) Quais são os principais benefícios e limitações de adotar TDD em um ambiente corporativo de alta pressão?

**5. Soft Skills e Cenários Críticos**

a) Relate uma situação em que você precisou lidar com **pressão alta** em um contexto de produção (por exemplo, falha sistêmica). Como você reagiu e o que aprendeu?

b) Como você lida com conflitos técnicos dentro de um time quando há opiniões divergentes sobre arquitetura ou design?

### **PARTE 2 — DESAFIO PRÁTICO**

**Objetivo:** Validar a capacidade técnica, qualidade do código e organização estrutural.

#### **Desafio: API de Controle de Acesso Simples**

Crie uma **API RESTful** utilizando **Node.js (versão 18 ou superior)** com **NestJS, Express ou AdonisJS** que atenda aos seguintes requisitos:

#### **Requisitos Funcionais**

**Cadastro de Usuários**

Campos: id, nome, email, senha, perfil (admin, user).

**Autenticação**

Endpoint de login que retorna um **JWT** válido.

**Controle de Acesso**

Apenas usuários com perfil admin podem listar todos os usuários.

**Logs de Acesso**

Cada login deve registrar um evento (data/hora, usuário, IP) em uma coleção/tabela separada.

#### **Requisitos Técnicos**

* Banco de dados: **PostgreSQL** ou **MongoDB** (escolha um).
* Estrutura modular e com princípios de **DDD** ou **Clean Architecture**.
* Aplicar **TDD** em pelo menos uma parte crítica (ex: autenticação).
* Utilizar **Docker**.
* Documentar endpoints com **Swagger** ou README bem detalhado.

#### **🚀 Diferenciais (não obrigatórios)**

* Uso de Redis para cache de autenticação
* Mensageria simples com **RabbitMQ** (ex: log de auditoria).
* Pipeline simples de CI (GitHub Actions, por exemplo).

#### **Entrega**

* Enviar o link do repositório (GitHub/GitLab/Bitbucket) ou arquivo .zip.
* Incluir instruções de instalação e execução.