# 🍽️ Desafio Técnico – MesaFácil

## Objetivo

Imagine que você deve criar uma **API REST** para gerenciar **reservas de mesas em restaurantes**.

Essa solução deve ser executada na **nuvem** e promover as seguintes funcionalidades:

- Cadastrar novos restaurantes
- Cadastrar clientes
- Criar reservas de mesas para um restaurante em determinado dia e horário
- Cancelar reservas
- Listar reservas existentes

Para fins de exercício, a solução deve ser construída em **Java** no backend. Frameworks e bibliotecas são de livre escolha (desde que não infrinjam direitos de uso).

É importante que os restaurantes, clientes e reservas sejam **persistidos** e **não se percam com o restart da aplicação**.

O foco dessa avaliação é a **organização, clareza e boas práticas na implementação da API REST**.
A comunicação deve ser feita através de **mensagens JSON**, e o formato das mesmas fica a seu critério.

## Como proceder

Por favor, implemente sua solução no seu repositório GitHub.
Ao final, **notifique da conclusão** para que possamos analisar o código implementado.

Lembre-se de deixar todas as **orientações necessárias para executar o seu código**.

## Tarefas bônus

### 🧍 Tarefa Bônus 1 - Controle de usuários

- Criar cadastro de usuários (clientes e administradores)
- Apenas usuários **admin** podem:
    - Cadastrar restaurantes
    - Visualizar todas as reservas
- Usuários comuns podem:
    - Criar e cancelar suas próprias reservas

⚙️ **Tarefa Bônus 2 - Performance**

Imagine que sua aplicação possa ser usada em cenários com **milhares de reservas simultâneas**.
Ela deve se comportar de maneira performática nesses cenários.
Testes de performance são uma boa maneira de garantir e observar como sua aplicação se comporta.

🧩 **Tarefa Bônus 3 - Versionamento da API**

Como você versionaria a API da sua aplicação?
Que estratégia usaria?

## Dicas e observações

- Teste bem sua solução e evite bugs
- Não inicie o teste sem sanar dúvidas
- A aplicação será executada para testes — documente dependências externas
- Inclua instruções claras no `README`
- Utilize **boas práticas** de código e arquitetura
- Documentação da API (ex: Swagger) é um diferencial

## O que será analisado

- Simplicidade no design da solução (evitar overengineering)
- Organização e arquitetura do código
- Boas práticas (legibilidade, manutenibilidade, padrões)
- Possíveis bugs
- Tratamento de erros e exceções
- Explicação breve das escolhas técnicas
- Uso de testes automatizados
- Limpeza do código
- Documentação da API e do projeto
- Logs e mensagens claras
- Organização dos commits