🧪 Desafio Técnico – Cadastro de Pacientes

Desenvolva uma aplicação completa (frontend + backend) para realizar o **cadastro e edição de pacientes**. O foco está na qualidade do código, estrutura da aplicação, arquitetura, uso de boas práticas e cobertura de regras de negócio. Não é necessário implementar autenticação/autorização.

🎯 Requisitos funcionais

1. **Cadastro de paciente com os seguintes campos:**
   - ID (gerado automaticamente)
   - Nome
   - Sobrenome
   - Data de nascimento
   - Gênero (Seleção: Masculino, Feminino, Outro)
   - CPF (não obrigatório, mas se preenchido, deve ser válido e único)
   - RG
   - UF do RG (seleção de estados brasileiros)
   - Email
   - Celular
   - Telefone fixo
   - Convênio (selecionado de uma lista vinda da base de dados)
   - Número da carteirinha do convênio
   - Validade da carteirinha (mês/ano)

2. **Funcionalidades:**
   - Cadastrar novo paciente
   - Editar paciente existente
   - Listar pacientes cadastrados
   - **Não permitir exclusão física de pacientes da base de dados**
   - Implementar **exclusão lógica** através de um campo Ativo ou Status (Ex: Ativo/Inativo)
   - Validação de dados:
     - CPF: válido (se informado) e único na base
     - Celular ou telefone fixo: pelo menos um preenchido e válido
     - Email válido
     - Data de nascimento não pode ser futura
     - Outras validações que achar necessárias

3. **Prevenir cadastros duplicados com base no CPF** (caso preenchido).

💾 Banco de Dados

- Use **SQL Server** como banco de dados.
- Crie os scripts SQL para:
  - Criação do banco de dados
  - Tabelas: Pacientes, Convênios
  - Dados mock para a tabela de convênios (mínimo 5 registros)

📦 Tecnologias e Requisitos Técnicos

- Backend obrigatoriamente em **.NET Core (C#)**
- Frontend em **Angular**
- Uso de mascaras em campos que necessitam (CPF, Telefone, etc)
- Separação de responsabilidades em camadas (Controllers, Services, Repositories)
- API RESTful com documentação simples (Swagger ou README com rotas)
- Interface responsiva e usável

🛠️ Persistência de dados:

Obrigatório utilizar um dos dois:

- **Entity Framework Core**, com uso de migrations
- ou **Dapper**, com queries organizadas e scripts claros

✨ Diferenciais (não obrigatórios, mas valorizados)

- Utilização de **migrations** (se usar EF)
- Testes automatizados (unitários e/ou integração)
- Deploy local com Docker (docker-compose com app e banco)
- Feedback visual e mensagens de erro claras no frontend
- Interface limpa e acessível

📦 Entrega

- Repositório Git público (GitHub, GitLab, etc.)
- Incluir um README com:
  - Instruções de instalação e execução
  - Como rodar o banco e popular a base
  - Descrição da arquitetura usada