Desafio: Desenvolver uma API de Gerenciamento de Tarefas em C# e Subir no Git

O objetivo deste desafio é desenvolver uma API em **C#** que implemente o CRUD para gerenciamento de tarefas e, no final, subir o código no **GitHub** para compartilhamento.

Requisitos:

1. Backend (API)

- **Tecnologia**: Utilize **ASP.NET Core Web API**.
- **Funcionalidade**:
  - A API deve permitir as operações de CRUD para tarefas:
    - **Criar** uma nova tarefa.
    - **Ler** todas as tarefas ou uma tarefa específica pelo `Id`.
    - **Atualizar** uma tarefa existente.
    - **Deletar** uma tarefa.
  - Cada tarefa deve ter os seguintes campos:
    - `Id` (int, auto incrementado)
    - `Título` (string, obrigatório)
    - `Descrição` (string, opcional)
    - `Data de Criação` (DateTime, gerado automaticamente)
    - `Data de Conclusão` (DateTime?, opcional)
    - `Status` (enum: `Pendente`, `EmProgresso`, `Concluída`)
- **Validações**:
  - O campo `Título` deve ser obrigatório e ter no máximo 100 caracteres.
  - A `Data de Conclusão` não pode ser anterior à `Data de Criação`.
- **Banco de Dados**:
  - Utilize **Dapper ou Entity Framework Core** para persistência.
  - Banco de dados: **SQL Server**.

2. Frontend (à sua escolha)

Podendo ser: WEB (**JS, React**) ou Descktop (**WPF**)

- Deve consumir os endpoints da API.
- Criar uma interface visual para:
  - Listar todas as tarefas.
  - Criar, editar e excluir tarefas.
  - Filtrar tarefas por status (`Pendente`, `EmProgresso`, `Concluída`).

2. Arquitetura em Camadas

- **Camadas**:
  - **Domain**: Contém as entidades e regras de negócio.
  - **Application**: Contém as interfaces e classes de serviço para a lógica de aplicação.
  - **Infrastructure**: Implementação dos repositórios e comunicação com o banco de dados.
  - **Presentation (API)**: Camada de comunicação HTTP (Controllers).

3. Regras de Clean Code

- **Nomeação**: Métodos e variáveis devem ter nomes claros e autoexplicativos.
- **Métodos Pequenos**: Quebre métodos longos em menores, focados em uma única responsabilidade.
- **SRP (Single Responsibility Principle)**: Cada classe deve ter uma única responsabilidade.
- **SOLID**: Aplique os princípios SOLID (foco em SRP e DIP - Dependency Inversion Principle).
- **Tratamento de Erros**: Capture exceções e retorne respostas HTTP adequadas (status 400 para erros de validação, status 500 para erros de servidor, etc.).

5. Publicação no Git

- **Versionamento**: Suba o código no **GitHub**.
- **Instruções**: No repositório, adicione um arquivo `README.md` contendo:
  - Passos para configurar o projeto localmente.
  - Comandos para rodar a aplicação e os testes.
  - Informações sobre o banco de dados usado.

Instruções para Submissão

1. **Desenvolva** a API seguindo os requisitos e regras mencionadas.
2. **Teste** a aplicação localmente para garantir o correto funcionamento de todas as operações.
3. **Suba** o código para um repositório no **GitHub** ou outra plataforma de controle de versão.
4. **Compartilhe** o link do repositório Git após a conclusão.