**ESTRUTURA DO PROJETO**

A estrutura de pastas é de extrema importância para definirmos um padrão de arquitetura. Devemos sempre saber a função que cada classe deve ter e onde ela deve ser colocada. Por exemplo os Controllers, todos devem ficar na pasta Controllers, mesma coisa para Interfaces, todas devem ser colocadas na pasta Interface.

As pastas que criei até o momento seguem um padrão que EU gosto de trabalhar, onde eu atribuo a cada classe unicamente sua funcionalidade. Seguem algumas explicações abaixo:

* **Controllers**: É onde todos os serviços externos irão bater para consumir nossa API. É onde escreveremos cada método que alguém de fora poderá consumir, e é nesses métodos das controllers que eles irão bater. Porém devemos manter na Controller apenas o necessário para receber um Request (Solicitação externa) e devolver um Response (Resposta para quem consumiu), para que as controllers fiquem o mais limpo possíveis. Quem irá resolver e cuidar de toda a lógica para tratar o resultado a ser devolvido serão as Services.
* **Services:** É a classe responsável por cuidar da maior parte da lógica de programação e tratar os resultados para devolver para a Controller. A Controller irá receber um request e irá chamar a service para resolver isso.
* **Repositórios:** É onde criaremos todos os métodos que executam querys no banco de dados. Por exemplo, caso a service precise bater no banco para obter os dados de algum cliente, ela deve chamar um método criado no repositório que irá executar essa query e devolver os resultados da base.
* **Interfaces:** Onde colocaremos a assinatura (nome) de todos os métodos de Services, Repositórios e etc (exceto de Controllers). Por exemplo, se você possuir uma service chama ClienteService, você deve obrigatoriamente criar uma Interface chamada IClienteService e colocar a assinatura de todos os seus métodos lá. Por convenção, as interfaces devem possuir o mesmo nome do que elas implementam, com a letra i maiúscula no inicio.
* **Models:** Onde criaremos modelos com parâmetros tipados. Por exemplo, se você deseja receber um json no request de um método da controller com os parâmetros “Documento” e “DataNascimento”, você deve criar uma classe com o nome que você quiser seguido de Request, e criar nela as propriedades Documento e DataNascimento. Ao instanciar uma nova variável com o tipo do modelo criado, ela herdará as propriedades criadas.
* **Configurations:** É onde configuraremos algumas funcionalidades uteis que o sistema irá possuir, como injeção de dependência e swagger.

**Swagger:** É um ambiente que facilitará muito os testes de mesa da API, pois ele listará todos os endpoints (métodos do controller) que podem ser consumidos, já dando exemplo de requisição.

**Injeção de Dependência:** É necessário informar o sistema quando uma interface implementa uma classe, e registrar isso se chama injeção de dependência. Isso causa uma grande alta na performance pois não será mais necessário instanciar classes inteiras para utilizar seus métodos.

* **Utils:** Onde colocaremos utilidades do sistema, como validações.
* **Dto:** São models que irão replicar as tabelas da base de dados. Por exemplo, se você possuir uma tabela chamada Cliente com as colunas ClienteId e Nome, você terá uma classe chama Dto, segundo o padrão de model, com os parâmetros ClienteId e Nome.