Controle			
Versão	**Data**	**Autor**	**Notas da Revisão**
1.0	16/09/2024	Ronaldo Costa Miranda	

Equipe		Período de:	**/**/**	Até:	**/**/**

Objetivos deste documento

Direcionar a análise da implementação da API .Net Core APITaskManager.

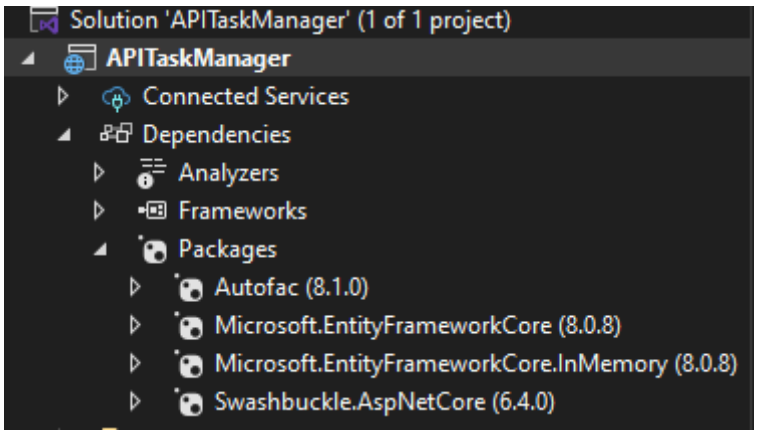
Cenário:

Criar uma API RESTful para gerenciar tarefas. A API deve permitir a criação, leitura, atualização e exclusão de tarefas. Usando o banco de dados em memória do Entity Framework Core.

Requisitos Funcionais:

Instalação prévia de um IDE (Integrated Development Environment) que possa abrir um projeto do tipo API .Net Core. (Vs ou Vs Code).
E a instalação dos packages:
Autofac
EntityFrameworkCore
EntityFrameworkCore.InMemory
O Swashbuckle.AspNetCore é instalado automaticamente na criação do projeto.

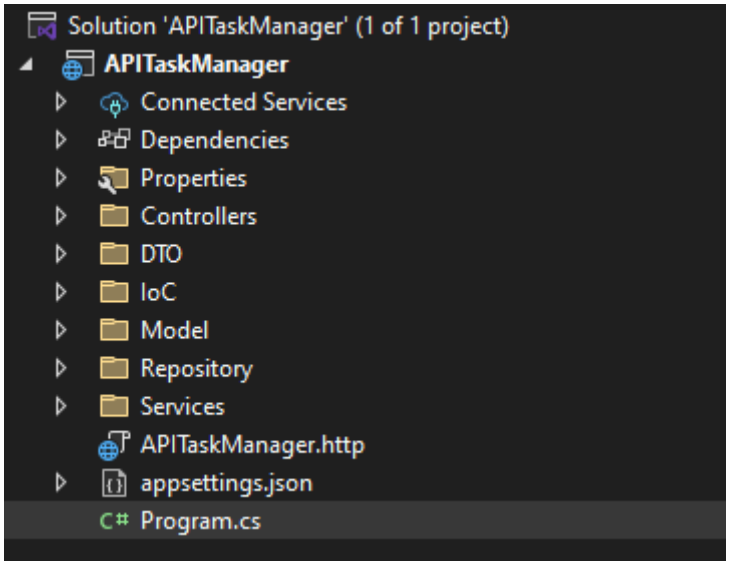
As versões utilizadas seguem a imagem abaixo.



Descrição: APITaskManager

Para cumprimento do desafio técnico de criar um Sistema de Gerenciamento de Tarefas criei a APITaskManager, com a ideia de ter uma API que realizasse as premissas de inclusão, edição e exclusão de tarefas, utilizando o Swagger como documentação e testes.

O projeto segue a seguinte estrutura:



Onde separei as responsabilidades por diretórios.

Controllers = Implementação dos controllers onde as requisições externas serão recebidas.

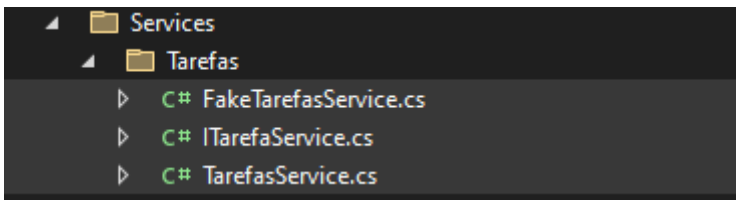
DTO – Data Transfer Objects => Para definir as classes de comunicação. A implementação Global permitirá a criação de subdiretórios a medida que o projeto evoluir. Por hora, tenho apenas a classe TarefaDTO que implementa a estrutura de comunicação com os Endpoints.

IoC – Inversion of Control – Foi a tentativa de implementar a Inversão de controle, para permitir o registro dos serviços e suas dependências (para possibilitar a injeção de dependências e seu cambiamento) na mesma implementação.

Model – Estruturação da classe e seus requisitos para a manipulação dos serviços da Tarefa e ações com o repositório.

Repository – Diretório que contem a construção do DbContext da Tarefa.

Services – Diretório onde foram implementadas as classes:



Classe FakeTarefasService – Seria a classe em que expandida a aplicação, poderia servir um controle Fake, que implementa a mesma interface de serviços ITarefaService que a classe oficial (de produção) TarefasService.

Classe ITarefaService – Define as implementações, ações que são obrigatórias para as classes que a herdam.

TarefaService – Seria a implementação oficial dos serviços disponíveis no Controller.

Com essa descrição da Estrutura a ideia era que estivesse disponível para requisições os métodos:

```
[HttpPost]
public async Task<IActionResult> CreateTarefa(TarefaDTO tarefaDto)...

[HttpPut("{id}")]
public async Task<IActionResult> UpdateTarefa(int id, TarefaDTO tarefaDto)...

[HttpDelete("{id}")]
public async Task<IActionResult> DeleteTarefa(int id)...
```

Para serem requisitados por chamadas externas vindas do controller TarefasController.

O construtor desse controller está preparado para inicializar os objetos de ITarefaService

```
[ApiController]
[Route("[controller]")]
public class TarefasController : ControllerBase
{

    private readonly ITarefaService _tarefaService;

    public TarefasController(ITarefaService tarefaService)
    {
        _tarefaService = tarefaService;
    }
```

Podendo receber e inicializar outras implementações de outras interfaces que possam vir a serem implementadas.

Pontos de Observações:

Banco de dados em memória do Entity Framework Core.
Essa funcionalidade permite que você utilize um banco de dados totalmente funcional dentro da memória do seu aplicativo
Particularmente, sabia que existia, mas nunca havia utilizado. As APIs as quais eu dou manutenção são conectadas diretamente com um banco de dados relacional e as poucas experiências que tive criando APIs, geralmente, começo criando um diretório de entidades, para criação da base de dados, antes de qualquer evolução, o que chamam de abordagem Model-First.

A maior dificuldade dessa implementação foi exatamente evoluir as implementações de Injeção de dependências e Inversão de controle, para cumprir os requisitos do desafio. De qualquer forma, mesmo não obtendo o êxito, posso acreditar que estava no caminho.

Como o projeto não pode ser finalizado, existem possibilidades de mudanças, a medida que novas implementações ocorrem, alguns arquivos ou diretórios podem surgir ou mudar de nó na estrutura.
O DTO, por exemplo é um diretório que pode se tornar subdiretório do Services, para indicar que apenas DTO's relacionados aos serviços de Tarefas estarão ali. Mas isso é razoável, caso novos subdiretórios de Services, além do Tarefas, surjam.

Não inseri da descrição, um exemplo de requisição, porque não cheguei ao teste de implementação.
Infelizmente o projeto será entregue sem a conclusão, apenas para que fique demonstrado o caminho que segui até aqui.

Considero um bom desafio, certamente o evoluirei até o final para ficar registrada a experiência de um projeto com conceitos que para mim são interessantes.

	Assinatura	**Data**
Analista de Desenvolvimento	Ronaldo Costa Miranda	16/09/2024