Introdução ao C# e .Net

Resumo

Para o desenvolvimento de aplicações, sejam web ou até mesmo *Console Applications*, é sempre necessário a escolha de qual tecnologia é a melhor para o atendimento da necessidade. Nesse trabalho não irá responder a essa pergunta, mas sim explicar e apronfundar sobre os conceitos básicos da linguagem de programação C# e da tecnoliga .Net. O objetivo é explicar os principais pontos dessa tecnologia e em quais meios ela é mais utilizada. Estamos falando de um resumo sobre o início e os princípios da tecnologia, então não acredite que esse trabalho o tornará um programador sênior.

Lista de ilustrações

Sumário

1 .Net - História e Conceitos

1.1 História

A Microsoft no final da década de 90 enfrentava um problema que muitas empresas enfrentam ainda hoje: uma gama de projetos e cada um com seus próprios padrões. Entretanto na Microsoft o problema estava nas linguagens que utilizava para o desenvolvimento de suas aplicações, uma vez que não havia muita definição em qual era mais apropriada. Assim, a reusabilidade de bibliotecas era quase impossível, assim como a reutilização de projetos para desevolvimento de novos. Quando se iniciava um novo projeto os programadores precisavam então não somente aprender tecnologias que já utilizavam, mas a nova para criação do projeto (ALURA, a).

Em meio a esse conflito foi escolhido o Java para iniciar o desenvolvimento dos novos projetos da Microsoft, que foi aceito inicialmente pelos engenheiros, mas que logo foi identificado certos problemas. O Java não se comunicava bem com as bibliotecas de código nativo que já existia, essas bibliotecas de código de máquina era importante para muitos projetos e acabou pesando bastante para a escolha de continuar ou não desenvolvendo com o Java.

Para resolução do problema, a Microsoft criou o projeto J++, contando com o engenheiro Anders Hejlsberg, que foi um dos principais nomes por trás do Delphi. O J++ foi uma linguagem Java que poderia apenas ser executada em ambiente windows, o que violava o contrato da Microsoft com a empresas Sun (responsável pelo Java), levando a um dos processos mais conhecidos da época (ALURA, a).

Sem a opção do J++ a Microsoft teve que se redezenhar para elaborar uma forma de centralizar os padrões dos seus projetos. Então foi iniciado um novo projeto que seria a base de todas suas soluções, que posteriormente foi chamado de **.Net**. A ideia desse novo ambiente de desenvolvimento foi trabalhar com diversas linguagens de programação, assim todas essas linguagens conseguiriam trabalhar em cima dos mesmos conjuntos de bibliotecas. Assim, a migração entre linguagens não precisaria preocupar com as bibliotecas finais, apenas com a lógica do sistema (ALURA, a).

Além da nova plataforma, a Microsoft também precisava de uma nova linguagem de programação, lançado em 2002 como projeto COOL (*C-like Object Oriented Languange*), a linguagem C# 1.0. Esse projeto reuniu conceitos de diferentes linguagens: Java, C, C++, Smalltalk, Delphi e VB. A linguagem foi lançada juntamente com o ambiente .Net 1.0, hoje conhecido como .Net Standard 1.0.

O tempo passou e hoje temos as novas versões do .net e da linguagem C# já consolidadas no mercado e utilizadas pelo mundo todo, as ferramentas de trabalho com a tecnologia

já é amplamente conhecida e com várias opções de adaptações entre plataformas e sistemas operacionais. Assim como as versões do .net também nasceu novas versões do ambiente de desenvolvimento.

1.2 .NET Standard

A Figura 1 a seguir lista as versões do .net Standard associadas com cada uma das versões de linguagens.

.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0	2.1
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	3.0
.NET Framework	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1[1]	4.6.1[1]	4.6.1[1]	*N/A*[2]
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4	6.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14	12.16
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8	5.16
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0	9.5
Unity	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	*TBD*
Universal Windows Platform	8.0	8.0	8.1	10.0	10.0	10.0.16299	10.0.16299	10.0.16299	*TBD*

Figura 1 – Evolução das versões .Net Standard

O Net standard é o ambiente de desenvolvimento que possibilita a criação de vários tipos diferenciados de sistema com diversas linguagens de programação. Fazendo ponte entre as plataformas com o objetivo de padronizar as bibliotecas de *low code*, o Net Standar não lança novas versões, mas mantém o susporte para a versão 2.1 (MICROSOFT,).

No surgimento da versão .Net5, não foi descontinuado o projeto .NET Standard, mas o novo framework carrga em si novas posibilidades de padrões de bibliotecas, possibilitando ainda que a comunicação entre bibliotecas antigas (principalmente do .net framework) seja respeitada na versão 2.1 do .NET Standard.

1.2.1 .Net Framework

O .Net Framwork executa por cima do .Net Standard e é possível criar diferentes tipos de projetos. Sua primeira versão possibilitava apenas sistema para desktop (windows), windows phone e aplicações web com o framework Asp.NET para ser hospedadas em servidores com Windows Server (ALURA, b). Com a chegada das aplicações embarcadas, o .net framework foi atualizado para dispobilizar outras formas de sistemas, principalmente com a inclusão do *xamarin*, tornando possível o desevolvimento de aplicações mobile, além do arduíno que possibilita a criação de aplicações embarcadas.

O .Net Framework se manteve por muais de décadas e ainda hoje há sistemas legados que o utiliza como base de desenvolvimento. Está atualmente na versão 4.8 e não será lançada nova versão para o mesmo. Vale ressaltar mais uma vez que essa versão não possibilita execução em outros sistemas operacionais senão o windows (salvo exceções), o que limita um pouco com o cenário de containers e multiplas possibilidade que temos no dia de hoje.

1.2.2 .Net Core

Acompanhando a evolução da tecnologia e abrindo os horizontes para novos sistemas operacionais, a Microsoft cria o .Net Core que é capaz de executar não somente no Windows. Sua criação marca uma evolução e na manutenção de dois sistemas ao mesmo tempo, o .Net Core e o .Net Framework. A Microsoft então por um certo tempo manteve os dois projetos afim de manter projetos legados com suporte à tecnologia, mas já deixar em aberto a possibilidade de criação de novos sistemas em uma tecnologia mais leve e robustas.

O .Net Core se evolui e chega na versão 3.1, quando chega a versão .NET 5, que unifica tanto o .net Framework quanto o .Net Core. A Figura 2 apresenta essa evolução de forma ilustrativa:

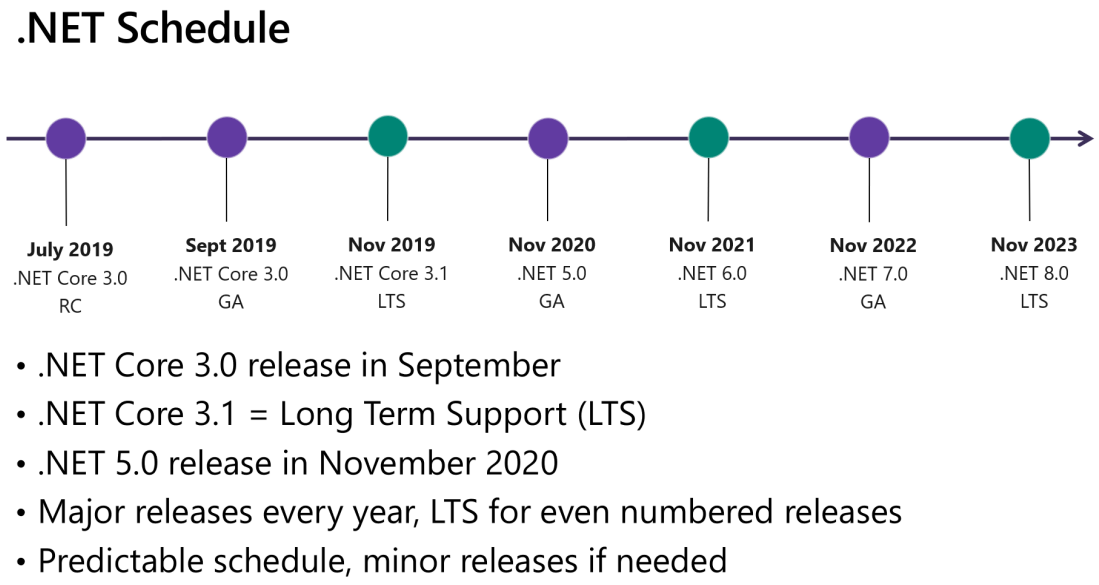

Figura 2 – Evolução das versões .Net Core

1.2.3 .NET

Com a chegada da nova versão para centralizar novamente as biliotecas de desenvolvimento, o .NET chega com a proposta de melhorar desempenho e de fornecer maior apoio aos desenvolvedores. Agora é possível criar vários tipos de projetos sem se preocupar com qual sistema operacional irá executar, mas ainda mantendo suporte aos sistemas legados antigos.

A evolução de cada uma das tecnologias marcam marcos tanto de melhora da tecnologia, mas também de evolução da linguagem e suas possibilidades. Várias bibiliotecas cores são melhoradas e ampliadas para atenderem mais solicitações.

1.3 Máquina Virtual - Common Language Runtime (CLR)

No processo de produção de um sistema existe os conceitos de compilação, interpretação e précompilação (máquina virtual). Várias linguagens, como o C e o Pascal, são compiladas, ou seja, o código é convertido para a liguagem de máquina, conforme o sistema operacional, muitas vezes sendo até limitado a apenas uma plataforma. Linguagens interpretadas são linguagens de tradução, basicamente uma interface interpreta o código e expõe o que foi interpretado (como o html e css trabalhando no navegador). O C# por outro lado carrega o conceito da máquina virtual, que é uma interface entre a aplicação e o sistema operacional, assim a pré compilação é feita para trabalhar por cima dessa interface, facilitando a exportação de plataformas e utilização de outras bibliotecas.

A figura 3 traduz bem o processo de compilação, onde o código é compilado para cada um dos sistemas operacionais.

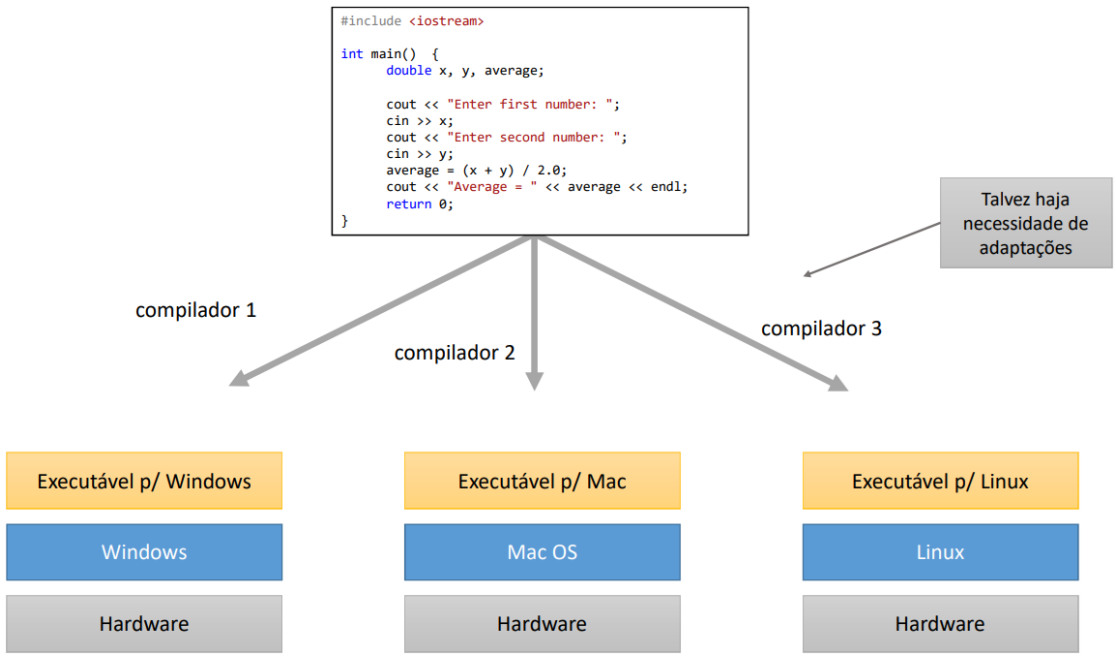

Figura 3 – Compilação para cada sistema operacional.

A figura 4 apresenta como é isolado o processo de pré compilação para execução no sistema operacional onde a aplicação é executada.

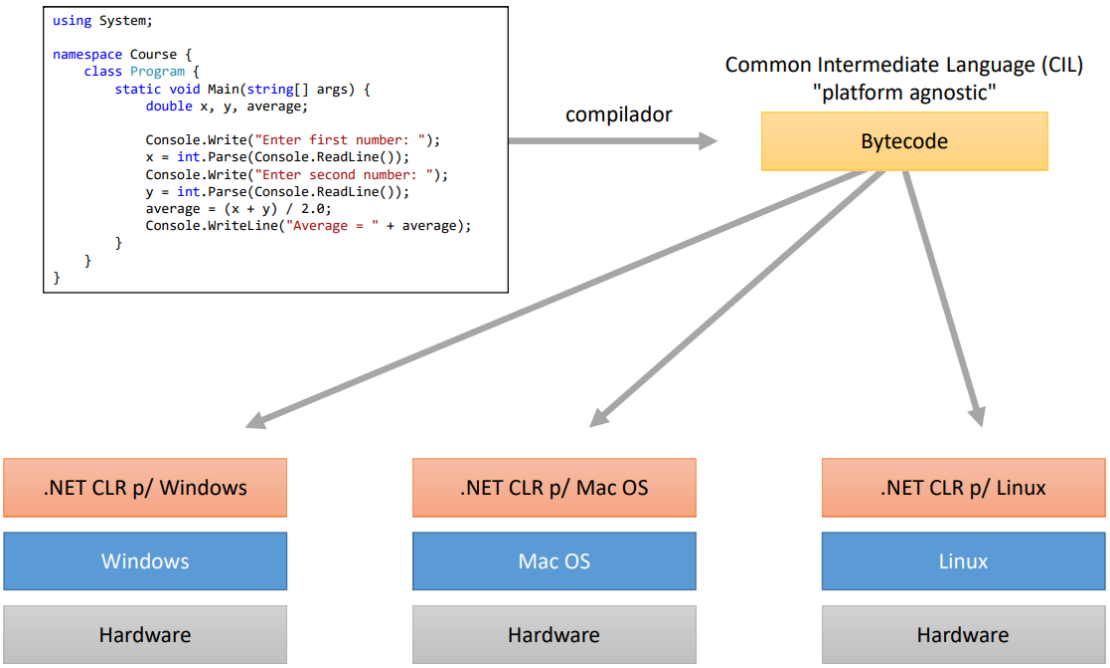

Figura 4 – Máquina virtual intermediando.

A CLR é o ambiente de execução para todas as linguagens da plataforma .Net, não apenas para o C#. Esse conceito não é novo, o Java já o utilizava para o mercado e já havia algumas linguagens com esses recursos, apesar de que serem encontradas mais no meio acadêmico (ALURA, a). Sua função é isolar a aplicação do sistema opercaional, dessa forma é fácil garantir a execução do sistema sem atrapalhar outros, como em uma queda brusta não derrubaria outros sistemas.

Como a máquina virtual pode trabalhar com diversas linguagens de programação diferentes, a CLR não executa diretamente o código do C#, ela executa uma linguagem intermediária comum a todas da plataforma .Net, a CIL (Common Intermediate Language). Para gerar o CIL que será executado pela CLR, precisamos passar o código C# por um compilador da linguagem, como o programa csc.exe. O compilador lê o arquivo com o código fonte do programa e o traduz para o código intermediário que será executado pela máquina virtual (ALURA, a).

2 Estrutura de uma aplicação C# .NET

O .Net é uma tecnologia que trás consigo o conceito de orientação a objetos, o que resulta no conceito de classes, que é basicamente o que compõe toda uma aplicação. O conjunto de classes e relacionamento entre elas trás as regras lógicas de um sistema. O conjunto dessas classes relacionadas compartilham um mesmo namespace, trazendo mais um nível de organização dentro do sistema.

O agrupamento de namespace trás uma organização lógica no sistema, ou seja, a organização física final não se separa por name space, afinal a saída do programa (exe ou dll) não é para cada classe ou namespace, e sim por cada projeto. A figura 5 apresenta como é feito a organização lógica e o porquê é importante segui-la.

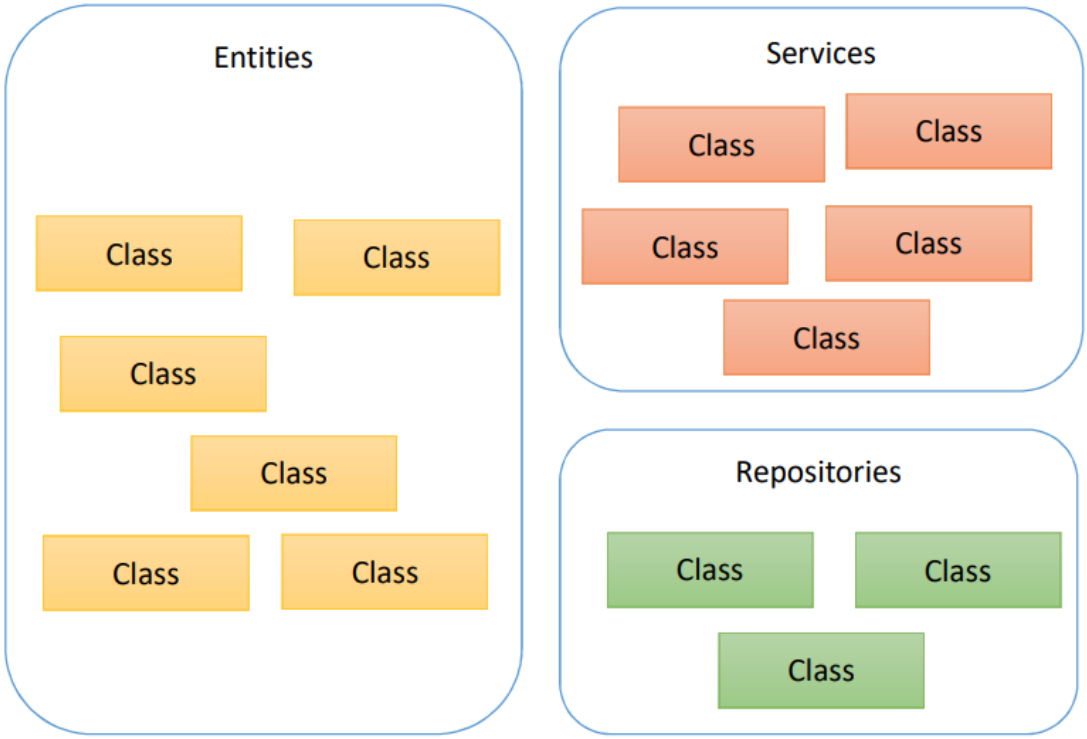

Figura 5 – Organização das classes por namespace.

A saída do projeto é conhecida como assembly pode ser uma dll ou um exe. Esses arquivos são agrupamentos físicos das classes relacionadas, assim como é ilustrado na imgaem 6.

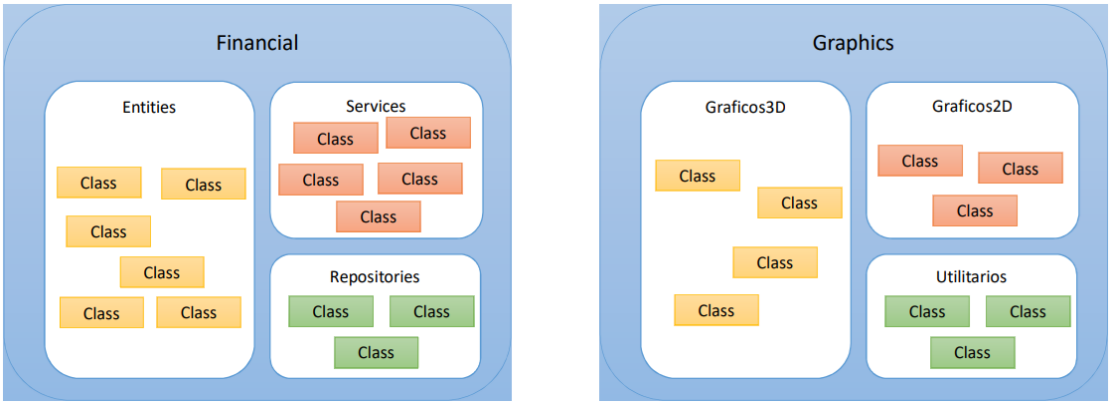

Figura 6 – Organização física das Classes relacionadas.

O conceito de aplicação vem do agrupamento de assemblies relacionados. Logo, uma aplicação é o conjunto de assemblies que comunicam entre si em função à uma execução em comum.

As aplicações .Net normalmente são organizadas então por Soluções, Projetos, Namespace e Classes. As Solução é o conjunto de Projetos que dão sentido ao sistema. Os projetos é o assembly em si, logo é o conjunto de namespaces relacionados. Os Namespaces são os conjuntos de classes que se relacionam. Por último a classe é uma tentativa de expressão de um objeto real que fornece sentido à execução de algo.

Ao criar um novo projeto, um Console por exemplo, sempre será criado uma Solução e um Projeto. Quando compilado o projeto ele executará naquele escopo da Solução, assim como se houver mais de um projeto dentro de uma mesma solução eles conseguem se referenciar.

Em muitos sistemas é comum dividir as reponsabilidades entre camadas, por exemplo a leitura de uma informação no banco de dados trás 3 responsabilidades diferentes: receber a requisição, tratar os dados de requisição, buscar os dados. Essas responsabilidades são comumente separadas em classes, mas o mesmo pode acontecer entre projetos. Então dentro do sistema (solução) que lê a informação no banco pode haver 3 projetos, o que recebe a requisição, o que trata e por fim o que busca os dados.

A Figura 7 exemplifica uma Solution que possui apenas um projeto, e nele possui outras classes que comunicam entre si. A classe controle rebe a solicitação, encaminha para a Servico que trata os dados, que chama a Repositori que trás as informações do banco de dados.

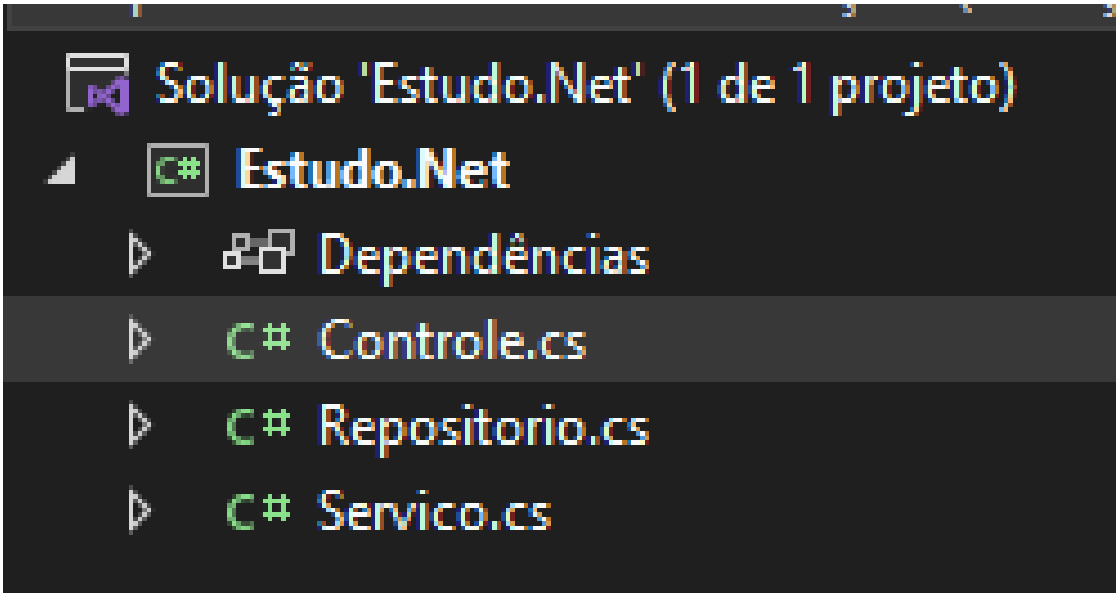

Figura 7 – Solução com 3 classes relacionadas

Uma forma de se organizar sem perder a lógica seria quebrando em três projetos diferentes, tudo dentro de uma mesma solution, apontando as dependências entre os projetos. A figura 8 apresenta esse exemplo, onde a lógica se mantém, mas a organização se difere.

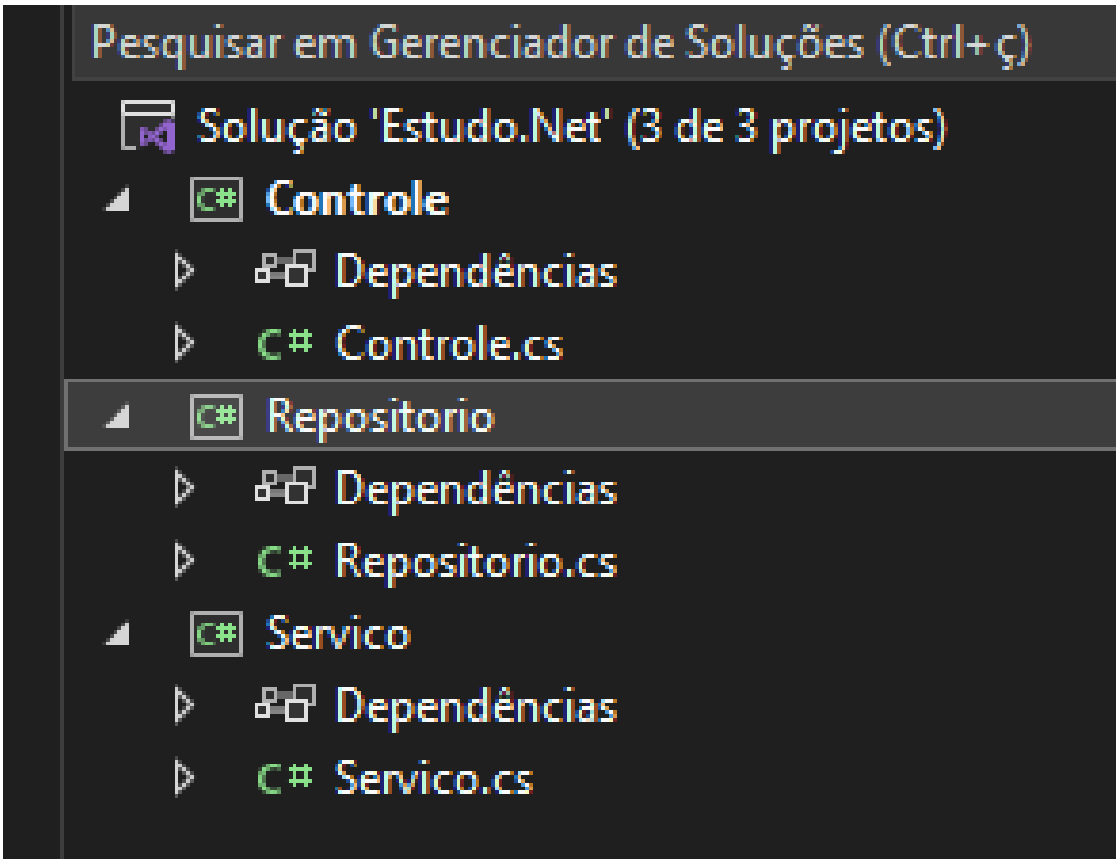

Figura 8 – Solução com 3 projetos relacionados

Referências

ALURA, L. *O QUE É C E .NET*. Disponível em: <https://www.alura.com.br/apostila-csharp-orientacao-objetos/o-que-e-c-e-net>. Citado 2 vezes nas páginas 5 e 9.

ALURA, L. *Tipos de aplicações possíveis com .NET*. Disponível em: <https://www.alura.com.br/artigos/o-que-e-net?utm_term=&utm_campaign=%5BSearch%5D+%5BPerformance%5D+-+Dynamic+Search+Ads+-+Artigos+e+Conte%C3%BAdos&utm_source=adwords&utm_medium=ppc&hsa_acc=7964138385&hsa_cam=11384329873&hsa_grp=169611649651&hsa_ad=703829337057&hsa_src=g&hsa_tgt=aud-409949667484:dsa-2276348409543&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gad_source=1&gclid=CjwKCAjwqMO0BhA8EiwAFTLgIN1pk1VumtxJnPc5PLR5miYOafpAA4rcqPb9Ps3zfftYCsFuFTYI5RoBwE>. Citado na página 6.

MICROSOFT, C. *.NET Standard*. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/standard/net-standard?tabs=net-standard-1-0>. Citado na página 6.