Arquitetura limpa com .NET Core: Introdução

POSTADO EM5 DE FEVEREIRO DE 2020POR JASON TAYLOR

Nos últimos dois anos, viajei pelo mundo ensinando programadores a construir aplicativos corporativos usando Clean Architecture com .NET Core. Comecei fornecendo uma solução de exemplo usando o icônico banco de dados Northwind Traders. Recentemente, desenvolvi um novo modelo de solução de arquitetura limpa para .NET Core.

Esta postagem fornece uma visão geral da Arquitetura Limpa e apresenta o novo Modelo de Solução de Arquitetura Limpa, um modelo de projeto .NET Core para criar aplicativos baseados em Angular, ASP.NET Core 3.1 e Arquitetura Limpa.

Vamos começar com uma visão geral da Arquitetura Limpa.

Visão geral

Com a Arquitetura Limpa, as camadas **Domínio** e **Aplicação** estão no centro do design. Isso é conhecido como o **núcleo** do sistema.

A camada de **domínio** contém a lógica e os tipos corporativos e a camada de **aplicativo contém a lógica e os tipos de negócios.** A diferença é que a lógica corporativa pode ser compartilhada em muitos sistemas, enquanto a lógica de negócios normalmente só será usada nesse sistema.

O núcleo não deve ser dependente de acesso a dados e outras preocupações de infraestrutura para que essas dependências sejam invertidas. Isso é obtido adicionando interfaces ou abstrações dentro **do Core** que são implementadas por camadas fora do **Core** . Por exemplo, se você quiser implementar o padrão Repository , faça isso adicionando uma interface no **Core** e adicionando a implementação em **Infrastructure** .

Todas as dependências fluem para dentro e o **Core** não depende de nenhuma outra camada. **Infraestrutura** e **Apresentação** dependem do **Core** , mas não uma da outra.

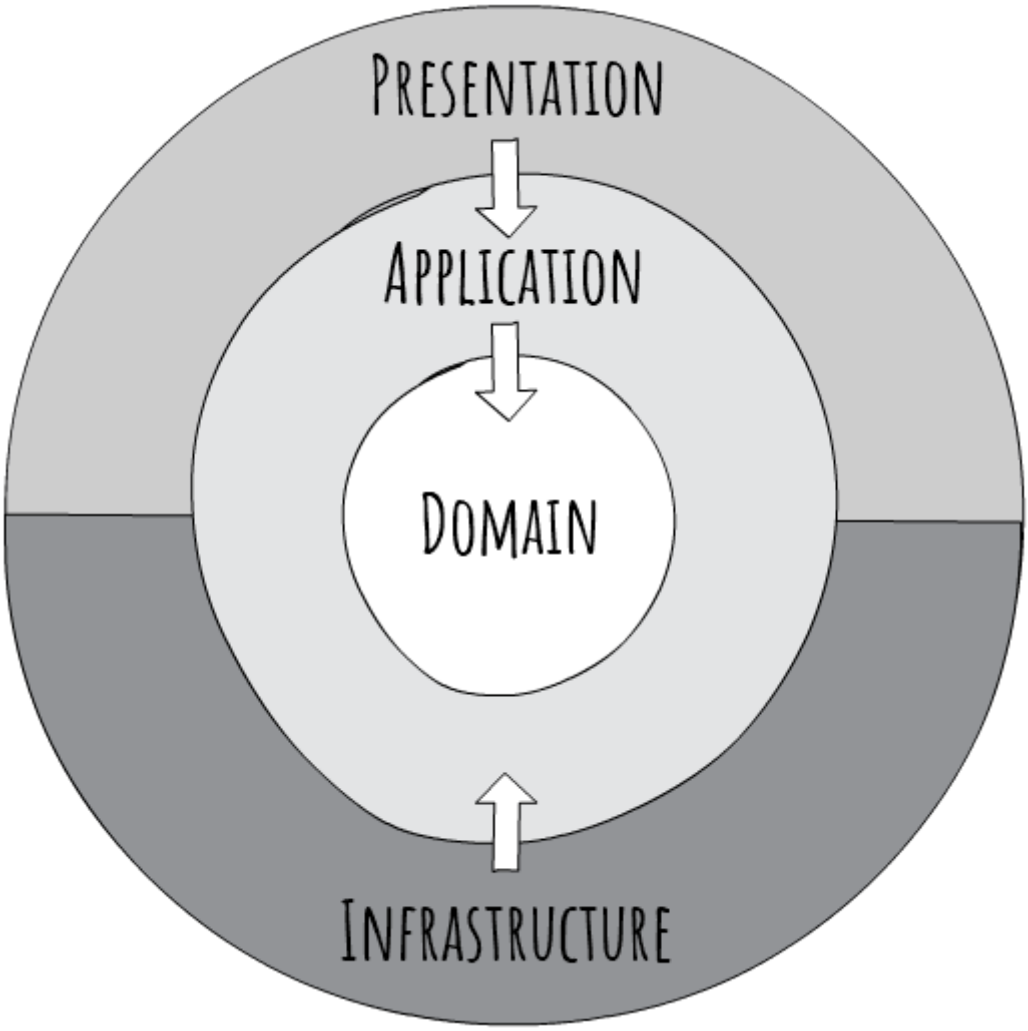


Figura: diagrama de arquitetura limpa

Isso resulta em arquitetura e design que é:

- **Independente de frameworks** não requer a existência de alguma ferramenta ou framework
- **Testável** fácil de testar – o **Core** não depende de nada externo, então escrever testes automatizados é muito mais fácil
- **A lógica independente da interface do usuário** é mantida fora da interface do usuário, por isso é fácil mudar para outra tecnologia - agora você pode estar usando Angular, em breve Vue, eventualmente Blazor!
- **Independentemente das** preocupações de acesso a dados do banco de dados, são separadas de forma limpa, portanto, a mudança do SQL Server para o CosmosDB ou de outra forma é trivial
- **Independente de qualquer coisa externa** , o **Core** é completamente isolado do mundo exterior – a diferença entre um sistema que durará 3 anos e um que durará 20 anos

No design acima, existem apenas três círculos, você pode precisar de mais. Pense nisso como um ponto de partida. Apenas lembre-se de manter todas as dependências apontando para dentro.

Vamos dar uma olhada em uma abordagem simples para começar com o novo modelo de solução de arquitetura limpa .

Modelo de solução

Este modelo fornece uma abordagem incrível para criar soluções baseadas em ASP.NET Core 3.1 e Angular 8 que seguem os princípios da Arquitetura Limpa. Se o Angular não é sua praia, não se preocupe, você pode removê-lo com facilidade. Nesta seção, você instalará o modelo, criará uma nova solução e revisará o código gerado.

Pré-requisitos

O primeiro passo é garantir que você atenda aos seguintes pré-requisitos:

- SDK do .NET Core (3.1 ou posterior)
- Node.js (6 ou posterior)

Verifique a versão do .NET Core executando este comando:

```
dotnet --list-sdks
```

Verifique a versão do nó executando este comando:

```
node -v
```

Em seguida, instale o modelo de solução usando este comando:

```
dotnet new --install Clean.Architecture.Solution.Template
```

Criar uma nova solução

Criar uma nova solução é fácil. Dentro de uma pasta vazia, execute o seguinte comando:

```
dotnet new ca-sln
```

A seguinte mensagem será exibida:

```
The template "Clean Architecture Solution" was created successfully.
```

Este comando criará uma nova solução, automaticamente com namespace usando o nome da pasta pai. Por exemplo, se a pasta pai for denominada **Northwind** , a solução será denominada **Northwind.sln** e o namespace padrão será `Northwind` .

A solução é criada usando o modelo de projeto Angular com ASP.NET Core. O projeto ASP.NET Core fornece um back-end de API e o projeto Angular CLI fornece a interface do usuário.

> **Observação**
> Leia Use o modelo de projeto Angular com ASP.NET Core para saber mais sobre essa abordagem.

Iniciar a solução do Visual Studio 2019 é trivial, basta pressionar **F5** .

Para iniciar a solução usando a CLI do .NET Core, são necessárias mais algumas etapas. Você pode saber mais visitando o link acima, mas vou incluir as informações aqui para completar.

Primeiro, você precisará de uma variável de ambiente chamada `ASPNETCORE_Environment` com um valor de `Development` . No Windows, execute `SET ASPNETCORE_Environment=Development` . No Linux ou macOS, execute `export ASPNETCORE_Environment=Development` .

Em seguida, execute o seguinte comando na pasta da solução:

```
cd src/WebUI
dotnet build
```

> **Observação**
> A compilação inicial levará alguns minutos, pois também instalará os pacotes do lado do cliente necessários. As compilações subsequentes serão muito mais rápidas.

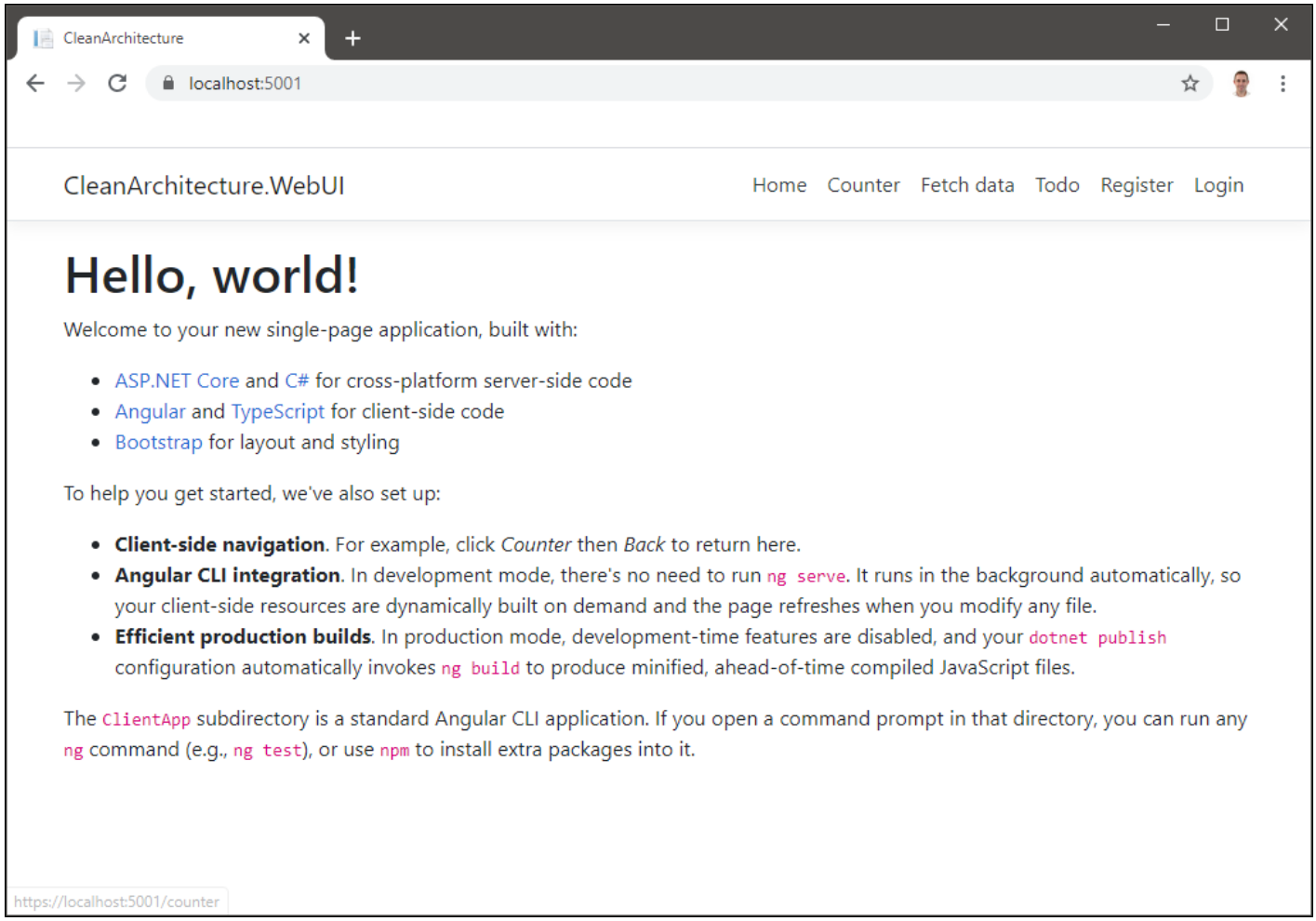
Em seguida, execute `dotnet run` para iniciar o aplicativo. A seguinte mensagem será exibida:

```
Now listening on: https://localhost:port
```

A porta geralmente é 5001. Abra o site navegando até https://localhost:port .

> **Observação**
> Você também verá uma mensagem semelhante à seguinte:
> *NG Live Development Server está escutando em localhost:port, abra seu navegador em http://localhost:port*
> Ignore esta mensagem, não é **a** URL para o ASP.NET Core combinado e Aplicação CLI angular

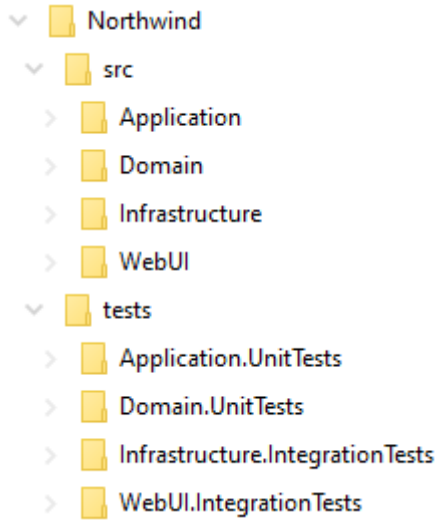
Se tudo deu certo, você verá o seguinte:



Vamos dar uma olhada na estrutura da solução recém-gerada.

Estrutura da solução

O modelo de solução gera uma solução de vários projetos. Para uma solução chamada **Northwind** , a seguinte estrutura de pastas é criada:



Os nomes de projetos em **src** se alinham de perto com as camadas do diagrama Clean Architecture, sendo a única exceção **WebUI** , representando a camada **Presentation .**

O projeto **Domain** representa a camada Domain e contém lógica corporativa ou de domínio e inclui entidades, enumerações, exceções, interfaces, tipos e lógica específica para a camada de domínio. Esta camada não tem dependências de nada externo.

O projeto **Aplicativo** representa a camada Aplicativo e contém toda a lógica de negócios. Este projeto implementa CQRS (Command Query Responsibility Segregation), com cada caso de uso de negócios representado por um único comando ou consulta. Essa camada é dependente da camada Domínio, mas não tem dependências de nenhuma outra camada ou projeto. Essa camada define interfaces que são implementadas por camadas externas. Por exemplo, se o aplicativo precisar acessar um serviço de notificação, uma nova interface será adicionada ao Aplicativo e a implementação será criada dentro da **Infraestrutura** .

O projeto **Infraestrutura** representa a camada Infraestrutura e contém classes para acessar recursos externos, como sistemas de arquivos, serviços da Web, SMTP e assim por diante. Essas classes devem ser baseadas em interfaces definidas na camada de Aplicação.

O projeto **WebUI** representa a camada Apresentação. Este projeto é um SPA (aplicativo de página única) baseado em Angular 8 e ASP.NET Core. Essa camada depende das camadas Aplicativo e Infraestrutura. Observe que a dependência da infraestrutura é apenas para dar suporte à injeção de dependência. Portanto , **Startup.cs** deve incluir a única referência à Infraestrutura.

Testes

A pasta de testes contém vários projetos de testes de unidade e integração para ajudá-lo a começar a trabalhar rapidamente. Os detalhes desses projetos serão explorados em um post de acompanhamento. Enquanto isso, sinta-se à vontade para explorar e fazer qualquer pergunta abaixo.

Tecnologias

Além do .NET Core, várias tecnologias são usadas nesta solução, incluindo:

- CQRS com MediatR
- Validação com FluentValidation
- Mapeamento de objeto-objeto com AutoMapper
- Acesso a dados com o Entity Framework Core
- API da Web usando ASP.NET Core
- IU usando Angular 8
- API aberta com NSwag
- Segurança usando ASP.NET Core Identity + IdentityServer
- Testes automatizados com xUnit.net , Moq e Shouldly

Nas postagens de acompanhamento, incluirei detalhes adicionais sobre como as tecnologias acima são usadas na solução.

Recursos adicionais

Neste post, forneci uma visão geral da Arquitetura Limpa e do novo modelo de solução. Se você quiser saber mais sobre qualquer um desses tópicos, dê uma olhada nos seguintes recursos:

- Modelo de Solução de Arquitetura Limpa
- Arquitetura limpa com ASP.NET Core 3.0 (NDC Sydney 2019)
- Regras para uma arquitetura mais limpa
- Excursão de superpoderes para desenvolvedores de arquitetura limpa
- Workshop de Arquitetura Limpa de 2 dias

Obrigado por ler. Por favor, poste quaisquer perguntas ou comentários abaixo.