**Extensões Úteis:**

- NuGet Gallery: Deixa o Visual Studio Code parecido com o Visual Studio para baixar as extensões úteis para nosso projeto. Ao usar Ctrl + Shift + P, você abre a ferramenta e consegue ver com mais facilidade o que está instalado e instalar novas funcionalidades.

**Ferramentas Úteis:**

- Microsoft.EntityFrameworkCore

- Microsoft.EntityFrameworkCore.Tools

- Microsoft.EntityFrameworkCore.Design

- Microsoft.EntityFrameworkCore.SqlServer

**GLOBAL.JSON**

O arquivo `global.json` é utilizado para definir a versão específica do SDK que desejamos utilizar em um projeto. Isso é especialmente útil quando há múltiplas versões de SDK instaladas na máquina.

**Formato do Arquivo:**

{

"sdk": {

"version": "5.0.102"

}

}

Como Criar o `global.json` via .NET CLI:

Para criar o arquivo `global.json`, utilize o seguinte comando no terminal:

dotnet new global.json --sdk-version 5.0.102

**Sobrescrevendo um `global.json` Existente:**

Se você deseja sobrescrever um arquivo `global.json` já existente, adicione a opção `--force` ao final do comando.

MIGRATIONS

O migrations no Entity Framework Core é uma ferramenta que permite gerenciar alterações no esquema do banco de dados de forma automatizada. Ele registra as alterações feitas nos modelos de dados e as traduz em comandos SQL para atualizar o banco de dados correspondente. Isso facilita o controle das mudanças na estrutura do banco de dados ao longo do desenvolvimento de um aplicativo, garantindo a consistência entre o modelo de dados e o esquema do banco de dados.

**Dotnet ef migrations list** – Vai listar todas as migrations disponíveis no projeto ou retornar nada caso não tenha migrations.

**Dotnet ef migrations add init -o Data/migrations** – Esse comando cria a migrations com base as classes modelos do seu projeto. **Dotnet ->** “invoca o dotnet” **ef -> “**Dentro do dotnet foi invocado o entity framework**” migrations ->** “dentro de ef foi invocado o migrations” **add ->** “Comando do migratiuons para adicionar uma migration” **init ->** ”o nome da migrations obs: pode ser qualquer nome” **-o Data/Migrations ->** “ esse comando cria a migrations dentro da pasta data e dentro da pasta migrations se as pastas não existirem ele cria as mesmas”

**Dotnet ef migrations remove** – Caso queira remover a migration criada.

**Dotnet ef database update** – Esse comando pega a migrations criada no projeto e sobe para um banco.

Camadas

Organizar um projeto em camadas oferece várias vantagens:

1. **Separação de Responsabilidades:** Cada camada tem um propósito específico, o que facilita a compreensão do código e a manutenção do sistema. Por exemplo, uma camada pode lidar com a lógica de negócios, outra com a apresentação da interface do usuário e outra com acesso a dados.

2. **Reusabilidade:** Camadas bem definidas facilitam a reutilização de código. Por exemplo, se a camada de acesso a dados é separada, ela pode ser usada por diferentes partes do sistema sem a necessidade de reimplementação.

3. **Escalabilidade**: Camadas modulares permitem que o sistema seja escalado de forma mais eficiente. Por exemplo, se a camada de interface do usuário precisar ser atualizada, isso pode ser feito sem afetar as outras camadas.

4. **Testabilidade**: É mais fácil testar unidades individuais do sistema quando elas estão em camadas separadas. Isso permite testes de unidade mais eficazes e também facilita a realização de testes de integração.

5. **Manutenção**: Com uma estrutura em camadas, é mais fácil realizar alterações ou atualizações em partes específicas do sistema sem afetar outras partes. Isso ajuda na manutenção a longo prazo do projeto.

Em resumo, organizar um projeto em camadas promove uma arquitetura mais limpa, modular e flexível, o que facilita o desenvolvimento, teste e manutenção do software.

No Visual Studio, temos o arquivo .sln que servirá para referenciar as camadas criadas. Já no VS Code, devemos criar esse arquivo manualmente. Por exemplo:

**dotnet new sln -n ProEventos**

Então devemos criar as camadas. Podemos fazer isso usando o comando `dotnet new classlib`, como mostrado no exemplo abaixo:

**dotnet new classlib -n ProEventos.Persistence**

**dotnet new classlib -n ProEventosDomain**

**dotnet new classlib -n ProEventos.Application**

Nesse modelo de camadas, a ideia é que a API entre em contato com a Application, que por sua vez utilize o Domain e acesse também a Persistência.

Próxima aula seção 6 aula 57 referenciando o projeto