**Seção 1:Introdução .NET Core**

**1 -- 2 Introdução**

**Para abrir o terminal:**

Abra o Visual Studio Code clique na aba superior em View => Terminal ou pelo atalho Ctrl + Shift + '

---------------------------------------------------

**Para saber a versão do dotnet. Digite o comando:**

No terminal digite. “dotnet --version”

---------------------------------------------------

**Para obter ajuda sobre os comandos:**

No terminal digite. “dotnet -help” ou “dotnet -h”.

---------------------------------------------------

**Para obter ajuda sobre um comando especifico:**

No terminal digite. “dotnet new -h”.

---------------------------------------------------

**Para limpar a lista de comandos no terminal.**

No terminal digite Ctrl + l.

---------------------------------------------------

**Para criar um Console:**

No terminal digite. “dotnet new Console -n [Nome]”

Obs: Todas as pastas e arquivos serão criados na pasta que o terminal estiver executando.

Obs: Navegue ate a pasta que deseja criar o arquivo atraves do comando cd.. ou cd [Nome da pasta].

---------------------------------------------------

**Executar um Console:**

dotnet run

Obs: Você deve estar na pasta do projeto.

Obs: Navegue ate a pasta que deseja atraves do comando cd.. ou cd [Nome da pasta].

---------------------------------------------------

**1 -- 4 Extensões dotnet**

**Visualizar todas as extensões instaladas:**

Clique no icone “Extensions” na barra ao lado esquerdo ou pressione Ctrl + Shift + X. Depois digite “@installed”.

---------------------------------------------------

**Instalar a extensão para desenvolver em C#:**

Clique no icone “Extensions” na barra ao lado esquerdo ou pressione Ctrl + Shift + X. Depois digite “C#” e clique sobre “install”.

---------------------------------------------------

**Instalar a extensão para facilitar a criação de classes e identificação de referencias em C#:**

Clique no icone “Extensions” na barra ao lado esquerdo ou pressione Ctrl + Shift + X. Depois digite “C# Extensions” e clique sobre “install”.

---------------------------------------------------

**Instalar a extensão para alterar a cor de metodos conhecidos:**

Clique no icone “Extensions” na barra ao lado esquerdo ou pressione Ctrl + Shift + X. Depois digite “Bracket Pair Colorizer” e clique sobre “install”.

---------------------------------------------------

**Instalar a extensão para facilitar a instalação de pacotes.**

Clique no icone “Extensions” na barra ao lado esquerdo ou pressione Ctrl + Shift + X. Depois digite “Nuget Package Manager” e clique sobre “install”.

---------------------------------------------------

**1 -- 5 Iniciando jogo**

**Como navegar ate a linha e coluna especifica:**

Pressione Ctrl + Shift + P. Depois digite “:numero da linha:numero da coluna”.

EX: “:9:29”

---------------------------------------------------

**1 -- 6 O Jogador**

**Para poder alterar mais de uma linha ao mesmo tempo crie varios cursores.**

Segure o ALT e clique sobre os lugares aonde voce quer criar outro cursor.

Depois so alterar o texto que voce quer.

---------------------------------------------------

**Criar outro cursor aonde tiver o texto igual.**

Selecione o texto e pressione CTRL + D, ele vai selecionar o primeiro texto igual. Continue pressionando CTRL + D para criar mais cursores.

---------------------------------------------------

**1 -- 8 A Interface**

Importante sempre criar uma interface para diminuir a dependencia sobre o objeto.

Ex: Remover o acomplamento sobre o jogador criando a interface ijogador.

---------------------------------------------------

**Para visualizar aonde o metodo esta declarado.**

Clique sobre o metodo com o botão direito => Peek => Peek Definition. Sera exibida uma janela com a declaracao do metodo.

---------------------------------------------------

**1 – 10 Várias Camadas**

**Para criar uma biblioteca:**

No terminal digite. “dotnet new classlib -n [Nome]”

Obs: Todas as pastas e arquivos serão criados na pasta que o terminal estiver executando.

Obs: Navegue ate a pasta que deseja criar o arquivo atraves do comando cd.. ou cd [Nome da pasta].

---------------------------------------------------

**Para adicionar a referencia de um projeto a outro projeto:**

No terminal digite. “dotnet add [NomeCompleto] reference [NomeCompleto]”

Ex: “dotnet add .\GameTop\GameTop.csproj reference .\GameTop.Lib\GameTop.Lib.csproj”

Obs: No projeto dentro do arquivo .csproj => ItemGroup => ProjectReference é possivel verificar ou adicionar as referencias.

---------------------------------------------------

**Para facilitar na localização do “NomeCompleto” do projeto:**

Digite a primeira letra do nome do projeto e pressione tab ate localizar a pasta do projeto, depois digite novamente a primeira letra do nome do projeto e pressione tab ate localizar o “.csproj”.

---------------------------------------------------

**Para criar uma solução:**

No terminal digite. “dotnet new sln -n [Nome]”

Obs: Utilizado no visual studio para abrir o grupo de projetos.

Obs: Todas as pastas e arquivos serão criados na pasta que o terminal estiver executando.

Obs: Navegue ate a pasta que deseja criar o arquivo atraves do comando cd.. ou cd [Nome da pasta].

---------------------------------------------------

**Para adicionar projetos a uma solução:**

No terminal digite. “dotnet sln [NomeCompleto] add [NomeCompleto]”

Ex: dotnet sln .\GameTop.sln add .\GameTop\GameTop.csproj

Obs: No projeto dentro do arquivo .sln => Project é possivel verificar ou adicionar as referencias.

---------------------------------------------------

**1 -- 12 Referências Using**

**Executar um restore na solução, igual ao rebuild do visual studio:**

No terminal digite. “dotnet restore”.

Obs: Navegue ate a pasta onte esta o .sln.

---------------------------------------------------

**Localizar erros:**

Na aba inferior clique sobre o X ou pelo atalho Ctrl + Shift + M

---------------------------------------------------

**Executar um projeto especifico:**

No terminal digite. “dotnet run --project [NomeCompleto]”.

---------------------------------------------------

**1 -- 13 Debug VS Code**

**Iniciar o Debug:**

Na aba lateral clique sobre o Run “Icone de um inseto” ou pelo atalho Ctrl + Shift + D,

depois na parte superior clique sobre Start Debugging.

Obs: Apos terminar a execução é necessario abrir o terminal e pressionar o enter para conseguir usar o terminal.

---------------------------------------------------

**Iniciar o dotnet watch:**

Watch serve para toda vez que um arquivo seja modificado e salvo o Visual Studio Code executa o projeto novamente.

No terminal digite. “dotnet watch run”.

Obs: Você deve estar na pasta do projeto.

Obs: Para parar o watch no terminal digite Ctrl + C ou Enter.

Obs: Se o AutoSave estiver habilitado a qualquer alteração fica rodando o sistema.

Obs: So executa apos uma alteração em um classe do projeto alvo. Não sera executado em caso de alteração em uma classe de um projeto que ele tenha referencia.

---------------------------------------------------

**Habilitar o AutoSave:**

Na aba superior clique sobre o File => Auto Save.

Obs: O sistema salva automaticamente qualquer alteração.

---------------------------------------------------

**1 -- 14. GIT**

**Iniciar o Git Hub:**

No terminal digite. “git init”.

Obs: Navegue ate a pasta onte esta o .sln.

---------------------------------------------------

**Para não enviar alguns arquivos para o Git Hub:**

Clique no botão New File e crie o arquivo “.gitignore”.

Obs: Dentro do arquivo coloque o nome das pastas ou arquivos que voce não que subir para o Git Hub.

Obs: Se o arquivo “.gitignore” for criado na raiz, os arquivos e pastas com nome igual ao declarado de todos os projetos serão ignorados.

---------------------------------------------------

**Criar arquivo sobre o projeto:**

Clique no botão New File e crie o arquivo “README.md”.

Obs: Dentro inserir informações sobre o projeto.

---------------------------------------------------

**Para dar um commit:**

No terminal digite. “git commit -m “[NomedoCommit]””.

Ou na aba lateral clique em Source Control atalho Ctrl + Shift + G, depois insira no campo mensagem o NomedoCommit e clique sobre Commit.

---------------------------------------------------

**Para enviar os dados:**

No terminal digite. “git push -u origin master”.

Ou na aba lateral clique em Source Control atalho Ctrl + Shift + G,

depois clique em “...” => Pull, Push => Push .

Ou na aba inferior ao lado do icone de erro no icone de “Synchronize Changes”.

---------------------------------------------------

**2 -- 15. Templates Web**

**Para criar um Projeto Web:**

No terminal digite. “dotnet new web -n [Nome]”

Obs: Projeto vai ser criado na mesma versão do SDK retornado pelo comando “--version”.

Obs: Todas as pastas e arquivos serão criados na pasta que o terminal estiver executando.

Obs: Navegue ate a pasta que deseja criar o arquivo atraves do comando cd.. ou cd [Nome da pasta].

---------------------------------------------------

**Para criar um Projeto MVC:**

No terminal digite. “dotnet new mvc -n [Nome]”

Obs: Projeto vai ser criado na mesma versão do SDK retornado pelo comando “--version”.

Obs: Todas as pastas e arquivos serão criados na pasta que o terminal estiver executando.

Obs: Navegue ate a pasta que deseja criar o arquivo atraves do comando cd.. ou cd [Nome da pasta].

---------------------------------------------------

**Para criar um Projeto WebAPI:**

No terminal digite. “dotnet new webapi -n [Nome]”

Obs: Projeto vai ser criado na mesma versão do SDK retornado pelo comando “--version”.

Obs: Todas as pastas e arquivos serão criados na pasta que o terminal estiver executando.

Obs: Navegue ate a pasta que deseja criar o arquivo atraves do comando cd.. ou cd [Nome da pasta].

---------------------------------------------------

**Diferença entre projetos Web, MVC e WebAPI:**

Web:

1. É o projeto web mais simples.
2. Não entende a arquitetura MVC.

WebAPI:

1. É o projeto web intermediário.
2. Entende a arquitetura MVC.
3. E trabalha com controllers.
4. Os retornos do controller default são JSON.

MVC: :

1. É o projeto web mais completo.
2. Entende a arquitetura MVC.
3. E trabalha com controllers, Models, Views.
4. Os retornos do controller default são Views.
5. Utiliza o razor, bootstrap.

---------------------------------------------------

**2 -- 16. Dotnet - Versões 2.2 e 3.1**

**Para obter ajuda sobre os comandos do dotnet:**

No terminal digite. “dotnet”.

---------------------------------------------------

**Para obter a lista dos SDKS:**

No terminal digite. “dotnet --list-sdks”.

---------------------------------------------------

**Para criar um projeto definindo a versão do SDK:**

Crie o arquivo global.json na pasta raiz dos seus projetos. Com o seguinte conteudo dentro:

{

    "sdk": {

        "version": [Com o numero completo da versão]

    }

}

Apos criar o arquivo, basta criar um novo projeto.

Obs: Todos os projetos criados na pasta vão seguir essa versão de SDK.

Obs: Caso seja informado um numero incorreto ao executar o comando “dotnet -- version” o terminal vai apresentar o seguinte erro “A compatible installed .NET Core SDK for global.json version [2.0.5] from [C:\Rafael\Udemy\Secao\_2\global.json] was not found”.

Obs: Caso tenha informado um numero correto ao executar o comando “dotnet -- version” o terminal vai apresentar a versão informada.

---------------------------------------------------

**2 -- 17. Inicio do Projeto**

**Configuração basica do projeto:**

Na classe Startup é possivel identificar a injeção de dependencia do objeto Iconfiguration.

Que no caso são os arquivos de configuração:

1. appsettings.json: Configurações do Ambiente de Produção.
2. appsettings.Development.json: Configurações do Ambiente de Desenvolvimento.

---------------------------------------------------

**Definir qual arquivo de configuração utilizar:**

Abra o arquivo “launchSettings.json” que está localizado dentro da pasta “Properties”. E altere o valor da propriedade “ASPNETCORE\_ENVIRONMENT” do Json para:

1. “Development” sera utilizado o arquivo “appsettings.Development.json”.
2. “Production” sera utilizado o arquivo “appsettings.json”.

---------------------------------------------------

**Desabilitar a abertura de um browser toda vez que executar o projeto:**

Abra o arquivo “launchSettings.json” que está localizado dentro da pasta “Properties”. E altere o valor da propriedade “launchBrowser” do Json para “false”.

---------------------------------------------------

**Para alterar a porta de execução do projeto:**

Abra o arquivo “launchSettings.json” que está localizado dentro da pasta “Properties”. E altere o valor da propriedade “applicationUrl” do Json definindo outra porta para o url.

---------------------------------------------------

**Desabilitar a execução do projeto em HTTPS:**

Abra o arquivo “launchSettings.json” que está localizado dentro da pasta “Properties”. E remova da propriedade “applicationUrl” o url com “https:”.

Abra a classe “Startup” e dentro do metodo “Configure” remova a linha:

“app.UseHttpsRedirection();”

---------------------------------------------------

**Como alterar parte do caminho de um controller:**

Abra a classe do controller e altere o: [Route("Com o caminho")].

Ex:

1. Antes “[Route("api/[controller]")]”
2. Depois [Route("site/[controller]")]

---------------------------------------------------

**2 -- 20. Data Context do EF Core**

**Instalando o SQLLite:**

No terminal digite. “Ctrl + Shift + p”. Sera aberto uma caixa para digitação:

1. Digite “>Nuget Package Manager: Add Package” e pressione enter.
2. Digite “Microsoft.EntityFrameworkCore.Sqlite” e pressione enter.
3. Selecione “Microsoft.EntityFrameworkCore.Sqlite” e pressione enter.
4. Selecione a versão.

Obs: Caso exista mais de um projeto no final sera pedido para selecionar o projeto que voce quer instalar o sqllite.

Obs: Necessario ter instalado a extensão “NuGet Package Manager”.

---------------------------------------------------

**Criar string de conexão:**

Abra a classe “appsettings.Development.json” e “appsettings.json” e insira dentro o seguinte conteudo:

"ConnectionStrings": {

"[ NomeDaStringDeConexao]": "Data Source=ProAgil.db"

},”.

---------------------------------------------------

**Definindo qual string de conexão utilizar:**

Abra a classe “Startup”, dentro do metodo “ConfigureServices” insira o seguinte conteudo:

“services.AddDbContext<DataContext>(d => d.UseSqlite(Configuration.GetConnectionString("[NomeDaStringDeConexao]")));”.

Obs: Para utilizar o “UseSqlite” pressione “Ctrl + .” e selecione “using Microsoft.EntityFrameworkCore;”

Obs: Caso a linha ““using Microsoft.EntityFrameworkCore;” não apareca:

1. É necessario ter instalado o SQLite antes. E possivel verificar se o SQLite esta instalado no arquivo “[NomeDoProjeto].csproj” se dentro existe a linha” <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="2.1.0"/>”
2. Caso esteja instalado e não apareca apenas execute o comando no terminal “dotnet build”.

---------------------------------------------------

**Criar conexão com o banco de dados utilizando o EntityFrameworkCore:**

Caso a pasta “Data” não exista crie a mesma. Dentro da pasta “Data” crie a classe “DataContext” que implementa a classe “DbContext”.

---------------------------------------------------

**Obter Objeto do EntityFrameworkCore:**

Dentro da classe “DbContext” crie a seguinte propriedade:

“public DbSet<[Objeto]> [Nome]{ get; set; }”.

---------------------------------------------------

**2 -- 21. Banco de Dados**

**Para obter ajuda sobre os comandos do EntityFramework:**

No terminal digite. “dotnet ef”.

---------------------------------------------------

**Para criar os arquivos de Migrations:**

No terminal digite. “dotnet ef migrations add [nome]”.

Obs: Se tudo estiver correto sera criado a pasta “Migrations”.

Obs: Atravez da classe que implementa o “Migration” é possivel visualizar o codigo para criação da tabela.

Obs: Se o seu objeto possuir uma propriedade int com o nome “[NomeDoObjeto]ID” sera criado uma PrimaryKey. Ex: “public int EventoID { get; set; }”

---------------------------------------------------

**Para criar uma tabela atraves da Migration:**

Dentro da classe “DbContext” crie a seguinte propriedade:

“public DbSet<[Objeto]> [Nome]{ get; set; }”.

Obs: Boa pratica sempre colocar o nome da propriedade no plural.

---------------------------------------------------

**Para criar um database atraves da Migration:**

No terminal digite. “dotnet ef database update”.

Obs: No diretorio que o terminal esta executando sera criado o arquivo .db com o nome que esta definido no “appsettings.Development.json” ou “appsettings.json” na “ConnectionStrings”.

Obs: Para visualizar a tabela utilize o “SQLite Database Browser”.

---------------------------------------------------

**Para atualizar um database atraves da Migration:**

Altere os seus objetos. Ex: Criando uma nova propriedade.

Depois no terminal digite:

1. “dotnet ef migrations add [nome]”.
2. “dotnet ef database update”.

---------------------------------------------------

**2 -- 22. Retornando Dados**

**Para recuperar os dados de um database atraves de um controller:**

1. Crie um construtor que receba como parametro o objeto “DataContext”.
2. Crie uma propriedade que receba o valor do “DataContext”.
3. Atraves da propriedade execute o ToList().

Ex: “return this.Context.Eventos.ToList();”

---------------------------------------------------

**2 -- 23. TryCatch e Status Code**

**Exibir Excecao sem tratamento de um controller:**

Caso o seu controller retorne uma “Exception”, se na classe “Startup” dentro do metodo “Configure” tiver a linha “app.UseDeveloperExceptionPage();”. Sera exibida uma pagina detalhando a excecao. Caso não possua essa linha sera exibida apenas “InternalServerError”.

Obs: Geralmente a linha “app.UseDeveloperExceptionPage();” so vem habilitada em modo de desenvolvimento.

---------------------------------------------------

**Remover o acoplamento do controller ao objeto:**

No retorno dos metodos ao inves de utilizar o “ActionResult<Objeto>” use o seguinte codigo:

“try

            {

                var result = this.Context.Eventos.ToList().FirstOrDefault(e => e.EventoID == id);

                return Ok(result);

            }

            catch(Exception ex)

            {

 return this.StatusCode(StatusCodes.Status500InternalServerError, "Banco de dados Falhou");

            }”

Obs: Você pode trocar o “return this.StatusCode()” por um “ return BadRequest();”. Mas nesse caso nao vai ser retornado erro 500.

---------------------------------------------------

**2 -- 24. Retorno Assíncrono**

**Para abrir uma nova thread a cada chamada no controller:**

1. Insira o “async” antes do resultado do metodo.
2. Troque o resultado do metodo para “Task<Objeto>”.
3. Insira antes do metodo assincrono o “await”.
4. Altere o metodo de recuperação de dados para assincrono.

Ex:

        public async Task<IActionResult> Get()

        {

            try

            {

                var result = await this.Context.Eventos.ToListAsync();

                return Ok(result);

            }

            catch

            {

                return this.StatusCode(StatusCodes.Status500InternalServerError, "Banco de dados Falhou");

            }

        }

---------------------------------------------------