**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 fechar um terminal:**

No terminal digite. “exit”.

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

**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.**

Pressione “Ctrl” + “Shift” + “Alt” e aperte a seta para cima ou para baixo.

Ou 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:

        public IActionResult Get()

        {

            try

            {

                var result = this.DataContext.Eventos.ToList();

                return Ok(result);

            }

            catch(Exception)

            {

                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");

            }

        }

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

**3 -- 27. Instalando e Rodando**

**Site para instalar o CLI do Angular:**

<https://angular.io/>

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

**Site para instalar o Node.js:**

<https://nodejs.org/en/>

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

**Para instalar o CLI do Angular:**

No terminal digite. “npm install -g @angular/cli”.

Obs: Para utilizar esse comando é necessario ter instalado o “Node.js”.

Obs: Caso ocorra o seguinte erro “*O termo 'npm' não é reconhecido como o nome de um cmdlet*” verifique:

1. Se ja está instalado o “Node.js”.
2. Reinicie o “Visual Studio Code”.

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

**Site para visualizar comandos do CLI do Angular:**

https://cli.angular.io/

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

**Criando um projeto em Angular:**

No terminal digite. “ng new ProAgil-App”.

1. Sera perguntado se você deseja criar a parte de rotas.
2. Digite “y” e pressione o enter.
3. Sera perguntado qual formato você quer utilizar.
4. Selecione o “CSS” e pressione o enter.

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 projeto em Angular:**

No terminal digite. “ng serve -o”.

Obs: “-o” Serve para abrir o browser.

Obs: Você deve estar na pasta do projeto.

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

**3 -- 28. Extensões Angular**

**Arquivo .gitignore dentro de um Projeto em Angular:**

Ao criar um novo projeto em Angular, automaticamente sera gerado um arquivo “.gitignore”.

Caso voce tenha um arquivo “.gitignore” na raiz dos seus projetos. Pegue o conteudo do “.gitignore” do projeto em Angular e insira no arquivo “.gitignore” que esta na raiz. Depois delete o que esta dentro do projeto em Angular.

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

**Instalar as extensões para Angular:**

* **Angular Files**

Para facilitar na criação de componentes, serviços e etc...

* **Angular Language Service**

Para conseguir acessar as informações dentro dos controllers.

* **angular2-switcher**

Para criar testes unitarios.

* **Auto Rename Tag**

Para mudar o nome da tag que fecha automaticamente quando alterar o nome da tag de abertura.

* **Bracket Pair Colorizer**

Para alterar a cor dos metodos conhecidos.

* **Debugger for Chrome**

Para conseguir debugar no Chrome.

* **npm = npm support for VS Code**

Para auxiliar na alteração do arquivo “package.json”.

* **Prettier - Code formatter**

Para formatar os arquivos seguindo as definições do arquivo “.editorconfig”.

* **TSLint**

Para auxiliar nos padrões de codificação.

Obs: Para instalar qualquer extensão, clique no icone “Extensions” na barra ao lado esquerdo ou pressione Ctrl + Shift + X. Depois digite o nome da extensão e clique sobre “install”.

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

**3 -- 29. Apresentando o Angular**

**Descrição dos arquivos em um projeto em Angular:**

* **Pasta e2e**

Pasta relacionada a testes unitarios.

* **Pasta node\_modules**

Pasta com todas as dependencias instaladas. Não precisa subir para o GitHub uma vez que o arquivo “package.json” reinstala todas caso a pasta não exista.

* **Pasta src**

Pasta raiz do projeto.

* **Arquivo .editorconfig**

Contem as configurações do editor. Utilizado pela extensão “Prettier - Code formatter”.

* **Arquivo angular.json**

Contem os assets, styles e os scripts.

* **Arquivo package.json**

Contem todas as dependencias do projeto. Que estão instaladas na pasta “node\_modules”.

Muito importante caso a pasta “node\_modules” seja excluida o sistema reinstala todas as dependencias automaticamente atraves desse arquivo.

* **Arquivo README.md**

Contem a descrição do projeto. Possivel visualizar atraves do GitHub.

* **Arquivo tsconfig.json**

Contem as configurações do projeto Angular.

* **Arquivo tslint.json**

Contem as configurações de padrões do java script.

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

**Descrição dos arquivos dentro da pasta src:**

* **Pasta app**

Contem os componentes do projeto.

* **Arquivo index.html**

Contem o conteudo da primeira pagina.

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

**Descrição de um Component:**

* **Import**

Todo componente em angular vai importar o “Component”:

import { Component } from '@angular/core';

* **Metadados**

São dados extras para o angular, definido com um decorator.

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

1. **selector:** Como componente sera utilizado no codigo html.
2. **templateUrl:** Qual é o template relacionado.
3. **template**: Codigo html do objeto. Ex: <h1>{{title}}</h1>.
4. **styleUrls**: Qual é o style relacionado.

* **Classe**:

São codigos para as views geralmente em Typescript. Toda classe possui Propriedades e Metodos.

export class AppComponent {

  title = 'ProAgil-App';

}

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

**Classe app.module.ts:**

Todo componente criado deve ser referenciado nessa classe.

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

**3 -- 30. Componentes**

**Do que compoem um Component:**

De um template, uma classe e metadados.

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

**Oque é um Template:**

É uma View ou um Layout criado em HTML. Inclui Binding e Diretivas.

Ex: Conteudo do objeto “app.component.html”.

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

**Criando um novo component:**

Clique sobre a pasta app e selecione “Generate Component”. Sera exibida uma janela para informar o nome do componente. Informe o nome e pressione “Enter”.

Obs: Ao criar o componente desta maneira ja serão criados os arquivos:

* **.css**
* **.html**

Conteudo do componente.

* **.spec.ts**

Arquivo de teste unitario.

* **.ts**

Arquivo principal do componente.

Obs: E o componente ja sera referenciado no arquivo “app.module.ts”.

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

**Criar uma propriedade dentro da Classe:**

[Nome]: [Tipo] = [Valor].

eventos: any = [

    {

      EventoID: 1,

      Tema: 'Angular',

      Local: 'Belo Horizonte'

    },

    {

      EventoID: 2,

      Tema: '.NET Core',

      Local: 'São Paulo'

    }

  ]

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

**Referenciar um componente a outro:**

Dentro do codigo html do componente insira “<Nome do Componente></Nome do Componente>”.

<app-eventos></app-eventos>

Obs: O nome do componente é definido na classe .ts nos metadados em “selector”.

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

**Utilizando uma propriedade no codigo HTML:**

No codigo HTML do componente digite {{Nome da propriedade}}.

{{eventos[0].Tema}}

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

**Utilizando uma lista de propriedades no codigo HTML:**

No codigo HTML do componente digite \*ngFor="let [nome do item] of [nome da lista]".

<div \*ngFor="let evento of eventos">

  Tema: {{evento.Tema}}

  <br>

  Local: {{evento.Local}}

  <hr>

</div>

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

**3 -- 31. Requests HTTP no Angular**

**Para habilitar uma chamada https atraves de um componente:**

Dentro da pasta “app” dentro do arquivo “app.module.ts” importe o conteudo da classe “HttpClientModule”.

import { HttpClientModule } from '@angular/common/http';

E adicione a classe “HttpClientModule” a lista de “imports”.

@NgModule({

  declarations: [

    AppComponent,

      EventosComponent

   ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    HttpClientModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

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

**Para realizar uma chamada https de um componente:**

* Dentro da classe do componente crie um construtor que receba atraves de injeção um “HttpClient”.

constructor(private http: HttpClient){}

Obs: Ao terminar de digitar o “HttpClien” e pressionar “Ctrl” + “Espaco” e deixar o mouse sobre “HttpClient”, o visual ja exibe uma mensagem informado que o HttpClient está disponivel como uma classe injetavel.

Performs HTTP requests. This service is available as an injectable class

* Crie um metodo para realizar a chamada https.

  getEventos(){

    this.http.get('http://localhost:5000/api/Values').subscribe(

      response => {

        this.eventos = response;

      }, error => {

        console.log(error);

      }

    );

* Realize a chamada do metodo antes da interfaze ser implementada atraves da implementação “OnInit”.

  ngOnInit(){

    this.getEventos();

  }

Obs: Caso o conteudo nao seja exibido clique sobre a pagina e clique em “Inspecionar” e verifique se existe erros na pagina.

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

**3 -- 32. CORS Angular e API**

**Caso ocorra o erro de “as been blocked by CORS policy: No 'Access-Control-Allow-Origin'”:**

É porque a sua aplicação não esta configurada para aceitar requisição cruzada. Para habilitar:

Dentro da classe “Startup.cs” dentro do metodo “ConfigureServices” adicione o seguinte conteudo:

services.AddCors();

Dentro da classe “Startup.cs” dentro do metodo “Configure” adicione o seguinte conteudo:

app.UseCors(x => x.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());

Obs: O “app.UseCors” deve ser colocado antes do metodo “app.UseMvc”.

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

**Para visualizar oque esta sendo retornado atraves do get para o componente:**

Insira o seguinte conteudo no metodo “console.log(response)”.

  getEventos(){

    this.http.get('http://localhost:5000/api/Values').subscribe(

      response => {

        console.log(response);

        this.eventos = response;

      }, error => {

        console.log(error);

      }

    );

  }

Obs: Ao abrir a pagina clique em “Inspecionar” depois selecione a aba “Console”.

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

**3 -- 33. Adicionando Bootstrap**

**Bootstrap:**

Site para download: <https://getbootstrap.com/>

Comando para instalar no terminal: “npm install bootstrap”

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

**Font Awesome:**

Site para download: <https://fontawesome.com/>

Comando para instalar no terminal: “npm install --save @fortawesome/fontawesome-free”

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

**Importando a referencia do Bootstrap e Font Awesome ao projeto:**

Dentro do arquivo “angular.json” localize a tag “styles” para identificar qual é a classe de css principal. Abra a classe principal e insira o conteudo:

@import '../node\_modules/bootstrap/dist/css/bootstrap-grid.min.css';

@import '../node\_modules/@fortawesome/fontawesome-free/css/all.min.css'

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

**Utilizar o Container do Bootstrap no componente:**

Altere a div do seu componente para “class=”container””:

<div class="container">

  <h1>

    Welcome to {{title}}

    </h1>

    <app-eventos></app-eventos>

</div>

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

**4 -- 35. Barra de Navegação**

**Obter exemplos de layout do Bootstrap:**

1. Entre no site: <https://getbootstrap.com/> e clique na aba “Examples”.
2. Clique sobre a miniatura do layout com o botão esquerdo.
3. Clique sobre o layout com o botão direito e “Inspecionar”.
4. Clique sobre o codigo com o botão direito e selecione “Copy” e “Copy element”.

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

**Organizar as linhas dos arquivos html:**

Digite. “Ctrl + Shift + P”. Sera aberto uma caixa para digitação, digite “>Reindent Lines” e pressione enter.

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

**4 -- 36. Grid dos Eventos**

**Criar um formulario simples no html:**

* Dentro do objeto html digite “form>.form-group.mb-2>input.form-control.mr-2”.

form>.form-group.mb-2>input.form-control.mr-2

* Pressione o enter

<form action="">

  <div class="form-group mb-2"><input type="text" class="form-control mr-2"></div>

</form>

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

**Criar um botão simples no html:**

* Dentro do objeto html digite “btn.btn.btn-outline-sucess{[Conteudo do botão]}”.

btn.btn.btn-outline-success{buscar}

* Pressione o enter

<button class="btn btn-outline-success">buscar</button>

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

**Criar uma tabela simples no html:**

* Dentro do objeto html digite “table.table.table-striped>(thead.thead-dark>tr>th\*7)”.

table.table.table-striped>(thead.thead-dark>tr>th\*7)

* Pressione o enter

<table class="table table-striped">

  <thead class="thead-dark">

    <tr>

      <th></th>

      <th></th>

      <th></th>

      <th></th>

      <th></th>

      <th></th>

      <th></th>

    </tr>

  </thead>

</table>

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

**Criar o body simples no html:**

* Dentro do objeto html digite “tbody>tr>td\*7”.

tbody>tr>td\*7

* Pressione o enter

  <tbody>

    <tr>

      <td></td>

      <td></td>

      <td></td>

      <td></td>

      <td></td>

      <td></td>

      <td></td>

    </tr>

  </tbody>

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

**4 -- 37. Binding e Interpolação**

**For no java script:**

Utilizando os elementos:

<tr \*ngFor="let evento of eventos">

Utilizando o indice:

<tr \*ngFor="let evento of eventos; let i = index">

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

**Utilizando binding entre o template e a classe:**

Para fazer um binding com uma propriedade de uma classe no codigo html digite “{{[Nome da propriedade]}}”.

<td>{{evento.tema}}</td>

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

**Diretivas extruturais no HTML:**

Sempre que tiver no codigo html \*ng[qualquer coisa] é uma diretiva extrutural. Porque altera a extrutura do seu template atraves de repetição ou condição.

<tr \*ngFor="let evento of eventos; let i = index">

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

**4 -- 38. ngIf**

**Aplicando condição para exibir conteudo:**

Sempre que tiver no codigo html \*ngIf=”[condição]” é uma diretiva extrutural para definir se exibe ou não aquele elemento.

<tbody \*ngIf="eventos && eventos.length">

Obs: Nesse caso esta verificando se o objeto eventos existe e se possui elementos.

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

**4 -- 39. Arquivos Estáticos**

**Habilitar o uso de arquivos estaticos:**

Dentro da classe “Startup.cs” dentro do metodo “Configure” adicione o seguinte conteudo:

app.UseStaticFiles();

Obs: Conteudo estatico como imagens ficam dentro da wwwroot.

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

**4 -- 40. Propriedade e Evento Binding**

**Exibindo uma imagem no html:**

        <img src="http://localhost:5000/img/{{evento.imagemURL}}"

        [style.width.px]="imagemLargura"

        [style.margin.px]="imagemMargem">

Obs: É possivel concatenar parte do caminho com uma propriedade.

Obs: Na largura e margem nao precisa utilizar {} para fazer referencia a uma propriedade.

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

**Declarando uma propriedade:**

Declarando informando o tipo.

imagemLargura: number = 50;

Declarando sem informar o tipo. Nesse caso é obrigatorio informar um valor.

imagemMargem = 2;

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

**Mover contudo digitado:**

1. Selecione o conteudo.
2. Pressione “Alt” + Seta para cima ou para baixo.

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

**Exibindo um botão no html:**

        <button class="btn btn-outline-primary"

        (click)="alternarImagem()">

          Mostrar imagem

        </button>

Obs: “alternarImagem” é um metodo da classe.

  alternarImagem(){

    this.mostrarImagem = !this.mostrarImagem;

  }

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

**4 -- 41. Two-way Data Binding**

**Tipo de conteudo html:**

* **Structural Directive**

\*ngIf="eventos && eventos.length"

* **Property Binding**

[style.margin.px]="imagemMargem"

* **Intepolation**

<td>{{evento.lote}}</td>

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

**Habilitando o Two-way Data Binding**

Dentro da pasta “app” dentro do arquivo “app.module.ts” importe o conteudo da classe “FormsModule”.

import { FormsModule } from '@angular/forms'

E adicione a classe “FormsModule” a lista de “imports”.

@NgModule({

  declarations: [

    AppComponent,

      EventosComponent,

      NavComponent

   ],

  imports: [

    CommonModule,

    BrowserModule,

    AppRoutingModule,

    HttpClientModule,

    FormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

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

**Utilizando o Two-way Data Binding:**

<form class="form-inline">

  <div class="form-group mb-2">

    Filtro: <input type="text"

    class="form-control mr-2"

    placeholder="buscar"

    [(ngModel)]="filtroLista">

    <button class="btn btn-outline-success">buscar</button>

  </div>

</form>

<h3>Filtrado por: {{filtroLista}}</h3>

Obs: [()] => Tambem é conhecido como caixa de bananas.

Obs: Quando for inserido um valor no “Filtro” o valor tambem sera exibido no “Filtrado por”.

Obs: Two-way data binding não funciona em um form simples.

<div class="form-inline">

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

**4 -- 42. Utilizando Pipes nos dados**

**Utilizando Pipe como mascara para os dados:**

Em qualquer interpolação você pode utilizar o “|” para alterar a forma de exibição do dado.

Alguns tipos:

**currency**: para moeda.

**lowercase**: deixa todo conteudo minusculo.

**uppercase**: deixa todo conteudo maiusculo.

**Json**: para exibir o arquivo json do objeto.

<td>{{evento.lote | uppercase}}</td>

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

**4 -- 43. Filtrando Grid**

**Propriedade vinculada ao Two-way Data Binding:**

Não é necessario ter uma propriedade para utilizar o two-way data binding. Ao utilizar o “ngModel” e declarar um nome, já é possivel utilizar esse nome de propriedade para fazer uma interpolacao.

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

**Encapsulando uma propriedade:**

Basta criar um get e um set para a propriedade.

  \_filtroLista: string;

  get filtroLista() : string {

    return this.\_filtroLista;

  }

  set filtroLista(value: string){

    this.\_filtroLista = value;

  }

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

**Selecionando alguns itens de uma lista:**

Utilize o “filter” como o “where” do C#.

return this.eventos.filter(e => e.tema.toLocaleLowerCase().indexOf(filtrarPor) !== -1);

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

**Criar um metodo com retorno:**

Declaração de um metodo [Nome]([Parametros]) : [Retorno].

  filtrarEventos(filtrarPor: string) : any {

    filtrarPor = filtrarPor.toLocaleLowerCase();

    return this.eventos.filter(e => e.tema.toLocaleLowerCase().indexOf(filtrarPor) !== -1);

  }

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

**5 -- 44. Solução, Domínio e Repositório**

**Criando novas camadas:**

Importante para dividir a responsabilidade em cada uma das camadas.

* **Domain:** Entidades e regras de negocio.
* **Repository**: Define como os objetos seram mantidos.
* **WebApi**: Camada de midwear.

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

**Remover arquivo ou pasta por linha de comando:**

No terminal digite. “rm -R [caminho completo do objeto]”.

rm -R .\ProAgil.Domain\Class1.cs

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

**5 -- 45. Referenciando Projetos**

**Executar um build:**

No terminal digite. “dotnet build”.

dotnet build

Obs: Caso o terminal esteja na raiz dos projetos e exista um sln, ira executar o build na solução. Ou seja vai executar o build em todos os projetos referenciados no sln.

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

**Referenciando WebAPI na solucao:**

Não é possivel refenciar um projeto WebAPI no sln porque ele não é um projeto dotnet ele é um projeto de interface.

Obs: É possivel criar no visual studio um webclient e colocar a interface dentro.

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

**5 -- 48. Novo Contexto**

**Declarar uma Foreign Key:**

Para criar uma chave estrangeira de outro objeto, por convenção o Entity Framework entende “[Nome do objeto]Id” é uma Foreign Key.

public int EventoID { get; set; }

Obs: Caso não siga a convenção pode ser criado uma nova propriedade e o Entity Framework entender que ela é a chave estrangeira.

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

**Instalando a referencia ao Entity Framework:**

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” e pressione enter.
3. Selecione “Microsoft.EntityFrameworkCore” e pressione enter.
4. Selecione a versão.
5. Selecione o projeto que deseja adicionar a referencia.

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

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

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

**Especificando uma chave composta:**

Na classe que implementa o “DbContext” sobrescreva o metodo “OnModelCreating”. Declarando quais são os ids da chave composta para o objeto.

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<PalestranteEvento>()

.HasKey(p => new { p.EventoID, p.PalestranteID });

}

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

**5 -- 49. Recriando Banco de Dados**

**Criando as migrations em uma biblioteca:**

Ao executar o comando “dotnet ef migrations add init” o visual retorna o erro “Startup project '[Nome do Projeto]' targets framework '.NETStandard'.” para informar que você não pode usar como alvo um projeto “.NETStandard”.

Para instalar basta executar esse comando “dotnet ef --startup-project [caminho do projeto] migrations add init”, informando o caminho do projeto.

Ex: “dotnet ef --startup-project ..\ProAgil.WebAPI migrations add init”.

Obs: Você deve estar na pasta do projeto e ter instalado a referencia do SQLite no projeto.

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

**Criando ou atualizando a base de dados**

Ao executar o comando “dotnet ef database update” o visual retorna o erro “Startup project '[Nome do Projeto]' targets framework '.NETStandard'.” para informar que você não pode usar como alvo um projeto “.NETStandard”.

Para criar ou atualizar basta executar esse comando “dotnet ef --startup-project [caminho do projeto] database update”, informando o caminho do projeto.

Ex: “dotnet ef --startup-project ..\ProAgil.WebAPI database update”

Obs: Você deve estar na pasta do projeto e ter instalado a referencia do SQLite no projeto.

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

**Coluna aceitando valor null:**

Ao definir um propriedade com “?” a frente do tipo. Ao criar as migrations essa propriedade aceitar valor nulo.

* **Propriedade**

public int? EventoId { get; set; }

* **Migration**

EventoId = table.Column<int>(nullable: true),

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

**5 -- 50. Projeto Repositório – Interfaces**

**Criar interface IProAgilRepository:**

Importante para remover a dependencia da sua aplicação para o ProAgilContext.

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

**5 -- 51. Repositório - Classe - Parte 01**

**Adicionar um item no banco de dados:**

Atraves da classe que implementa o “DbContext” chame o metodo “Add”.

this.ProAgilContext.Add(entity);

Obs: Necessario executar o comando “SaveChanges” ou ““SaveChangesAsync”” para operação ser executa no banco de dados.

Obs: Salva todos os objetos filhos tambem.

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

**Atualizar um item no banco de dados:**

Atraves da classe que implementa o “DbContext” chame o metodo “Update”.

this.ProAgilContext.Update(entity);

Obs: Necessario executar o comando “SaveChanges” ou ““SaveChangesAsync”” para operação ser executa no banco de dados.

Obs: Salva todos os objetos filhos tambem.

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

**Remover um item no banco de dados:**

Atraves da classe que implementa o “DbContext” chame o metodo “Remove”.

this.ProAgilContext.Remove(entity);

Obs: Necessario executar o comando “SaveChanges” ou ““SaveChangesAsync”” para operação ser executa no banco de dados.

Obs: Salva todos os objetos filhos tambem.

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

**5 -- 52. Repositório - Classe - Parte 02**

**Retornar os objetos dentro do objeto atraves do EntityFramework:**

Execute o metodo “Include” para retornar os objetos e “ThenInclude” para retornar os objetos dentro do objeto.

        if(includePalestrantes)

            {

                query = query

                    .Include(pe => pe.PalestranteEventos)

                    .ThenInclude(p => p.Palestrante);

            }

Obs: So utilizar o “ThenInclude” caso voce queira o “Palestrante” dentro do “PalestranteEventos”.

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

**Utilizando o IQueryable:**

Utilizando o “IQueryable” para definir oque e como serão retornados os dados do banco de dados.

public async Task<Evento[]> GetAllEventoAsync(bool includePalestrantes = false)

        {

            IQueryable<Evento> query = this.ProAgilContext.Eventos

                .Include(e => e.Lotes)

                .Include(e => e.RedeSociais);

            if(includePalestrantes)

            {

                query = query

                    .Include(pe => pe.PalestranteEventos)

                    .ThenInclude(p => p.Palestrante);

            }

            query = query.OrderByDescending(e => e.DataEvento);

            return await query.ToArrayAsync();

        }

* **Include:** Retorna o objeto.
* **ThenInclude:** Retorna o objeto apos o include.
* **OrderBy:** Ordena a lista.

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

**5 -- 54. Controllers Evento**

**Utilizando injeção de dependencia para receber no costrutor da controller uma interface:**

Dentro da classe “Startup.cs” dentro do metodo “ConfigureServices” adicione o seguinte conteudo “services.AddScoped<[Interface], [Implementação]>();”

services.AddScoped<IProAgilRepository, ProAgilRepository>();

Dentro do controler no construtor receba a interface.

        public EventoController(IProAgilRepository iProAgilRepository)

        {

            this.IProAgilRepository = iProAgilRepository;

        }

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

**Receber um parametro no metodo da controller**

Dentro da “Controller” altere o identificador de ação para “[identificador(“{Parametros}”)]”.

[HttpGet("{eventoId}")]

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

**Alterar caminho de um metodo da controller**

Dentro da “Controller” altere o identificador de ação para “[identificador(“caminho{Parametros}”)]”.

[HttpGet("getByTema/{tema}")]

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

**Retornar objeto adicionado atraves da controller**

Dentro do metodo retorne o “Created” informando a url de consulta por id e o id do objeto.

return Created($"/api/evento/{model.Id}", model);

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

**5 -- 55. Testando Controller Evento**

**Looping infinito ao tentar obter um dado:**

Caso o seu objeto tenha uma chave estrangeira para outro objeto e você tenha criado uma propriedade para facilitar, essa propriedade deve ser readonly.

public int EventoId { get; set; }

public Evento Evento { get; }

Obs: Caso a propriedade Evento tenha o set ao tentar obter o dado o sistema entra em looping infinito.

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

**ApiController:**

Caso voce declare na sua controller o “ApiController” todos os metodos ja retornam erro de validação.

Ex: Data informando dia/mes/ano retorna “Could not convert string to DateTime”.

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

**5 -- 56. Tracker e No Tracker (EF Core)**

**Entity Framework travando objeto para atualização:**

Caso voce esteja utilizando uma consulta antes do comando de “Remove” ou “Update” o Entity Framework pode travar aquele objeto para ser atualizado.

Para impedir que o Entity Framework trave o objeto existem duas opções:

* **Forma especifica**

No metodo que retorna o objeto. Na consulta do objeto voce deve utilizar o metodo “.AsNoTracking()”.

query = query.AsNoTracking()

            .OrderByDescending(e => e.DataEvento);

return await query.ToArrayAsync();

Obs: Apenas esse metodo nao sera travado.

* **Forma geral**

No construtor ao obter a classe que implementa o “DbContext” set a propriedade “ChangeTracker.QueryTrackingBehavior” para “QueryTrackingBehavior.NoTracking”.

public ProAgilRepository(ProAgilContext proAgilContext)

{

    this.ProAgilContext = proAgilContext;

    this.ProAgilContext.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;

}

Obs: Dessa maneira nenhum metodo sera travado.

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

**6 -- 57. Interfaces Models**

**Criar interfaces em Typescript:**

1. Dentro da pasta “app” crie a pasta “\_models” caso não exista.
2. Clique com o botão direito do mouse na pasta “\_models” e selecione “Generate Interface”.
3. Informe o nome.

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

**Declarando uma propriedade em Typescript:**

Dentro da classe digite “[Nome da Propriedade]: [Tipo]”.

id: number;

local: string;

dataEvento: Date;

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

**Declarando uma propriedade que pode ser nula:**

Dentro da classe digite “[Nome da Propriedade]?: [Tipo]”.

dataInicio?: Date;

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

**Declarando uma lista em Typescript:**

Dentro da classe digite “[Nome da Propriedade]: [Tipo][]”.

lotes: Lote[];

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

**Alterar texto para minusculo:**

1. Selecione o texto.
2. Pressione “Ctrl” + “Shift” + “p”.
3. Digite “>Transform to Lowercase” e pressione o enter.

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

**Referenciando um objeto em Typescript:**

Na parte superior digite “import { [Nome do Objeto] } from "[Caminho do Objeto]";”

import { Lote } from "./Lote";

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

**6 -- 58. Evento Service - Parte 01**

**Criar um Service em Typescript:**

Importante para encapsular o acesso a um metodo.

Ex: Criar um serviço para encapsular o acesso a classe “HttpClient”. Apenas o serviço fara a conexao http para realizar um “Get” ou “Post”.

1. Dentro da pasta “app” crie a pasta “\_services” caso não exista.
2. Clique com o botão direito do mouse na pasta “\_services” e selecione “Generate Service”.
3. Informe o nome.

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

**Criar um metodo em um Service:**

Dentro do Service digite “[Nome do metodo]([Parametros]){[Conteudo]}”.

  getEvento(){

    return this.http.get(this.baseURL);

  }

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

**Receber o Service em um Component:**

Para utilizar o Service em um Component existem 3 opções:

1. **Maneira de qualquer Component no projeto ter acesso ao Service.**

Dentro do Service no decorator informe o “providedIn”.

@Injectable({

  providedIn: 'root'

})

1. **Outra maneira de qualquer Component no projeto ter acesso ao Service.**

Dentro do Service no decorator remova o “providedIn”.

@Injectable()

Dentro da classe “app.module.ts” importe o Service e dentro do “@NgModule” em “providers” insira o nome do Service.

import { EventoService } from './\_services/Evento.service';

providers: [ EventoService ],

1. **Maneira de apenas esse componente ter acesso ao Service.**

Dentro do Service no decorator remova o “providedIn”.

@Injectable()

Dentro do Component importe o Service e dentro do “providers” insira o nome do Service.

import { EventoService } from '../\_services/Evento.service';

@Component({

  selector: 'app-eventos',

  templateUrl: './eventos.component.html',

  styleUrls: ['./eventos.component.css'],

  providers:[ EventoService ]

})

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

**Recebendo o Service em um Component:**

* Importe o Service na classe em TypeScript.

import { EventoService } from '../\_services/Evento.service';

* No construtor da classe em TypeScript receba o Service como parametro.

constructor(private eventoService: EventoService){}

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

**6 -- 60. Observable**

**Retornando um objeto atraves de um metodo:**

Dentro do Service digite “[Nome do metodo]([Parametros]) : Observable<Objeto>{[Conteudo]}”.

  getEvento() : Observable<Evento[]>{

    return this.http.get<Evento[]>(this.baseURL);

  }

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

**Criar um template string:**

Digite `${Objeto}`.

`${this.baseURL}/getByTema/${tema}`

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

**6 -- 61. Ajustando Lotes e Imagems**

**Exibir um icone em um botao no html:**

Dentro do botão digite “<i class=”[Icone]”></i>”.

<i class="{{mostrarImagem ? 'fa fa-eye-slash' : 'fa fa-eye'}}"></i>

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

**6 -- 62. Criando filtro de Data- Pipes**

**Criar uma constante em Typescript:**

1. Dentro da pasta “app” crie a pasta “util” caso não exista.
2. Clique com o botão direito do mouse na pasta “util” e selecione “Generate Class”.
3. Informe o nome.

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

**Declarar uma variavel constante em Typescript:**

Dentro da classe digite “static readonly [Nome da Variavel] = [Valor]”.

static readonly DATE\_FMT = 'dd/MM/yyyy';

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

**Formatar um objeto em Typescript:**

Para formatar um objeto em Typescript sua classe Precisa herdar a classe “DataPipe”. Depois digite “super.transform([Objeto], [Formato])”.

export class DateTimeFormatPipePipe extends DatePipe implements PipeTransform {

  transform(value: any, args?: any): any {

    return super.transform(value, Constants.DATE\_TIME\_FMT);

  }

}

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

**Criando um Data-Pipe:**

1. Dentro da pasta “app” crie a pasta “\_helps” caso não exista.
2. Clique com o botão direito do mouse na pasta “\_helps” e selecione “Generate Pipe”.
3. Informe o nome.

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

**Habilitar um Data-Pipe para uso:**

Dentro da pasta “app” dentro do arquivo “app.module.ts” importe o conteudo do Pipe.

import { DateTimeFormatPipePipe } from './\_helps/DateTimeFormatPipe.pipe';

E dentro do “@NgModule” em “declarations” insira o nome do Pipe.

  declarations: [

    AppComponent,

      EventosComponent,

      NavComponent,

      DateTimeFormatPipePipe

   ],

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

**Utilizar um Data-Pipe em html:**

Dentro do codigo html digite “{{[Propriedade] | [Nome do Pipe]}}”.

<td>{{evento.dataEvento | DateTimeFormatPipe}}</td>

Obs: O nome do Pipe é o declarado dentro do Pipe em “name”.

@Pipe({

  name: 'DateTimeFormatPipe'

})

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

**6 -- 63. NGX Bootstrap**

**Site para instalar o NGX Bootstrap:**

<https://valor-software.com/ngx-bootstrap/#/>

Obs: Para saber como utilizar os componentes, no site existe uma bara que mostra todos os componentes e como utilizalos.

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

**Instalar o NGX Bootstrap:**

No terminal digite “npm install ngx-bootstrap --save”.

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

**Importando componentes do NGX Bootstrap:**

Dentro da pasta “app” dentro do arquivo “app.module.ts” importe os componentes.

import { ModalModule } from 'ngx-bootstrap/modal';

import { BsDropdownModule } from 'ngx-bootstrap/dropdown';

import { TooltipModule } from 'ngx-bootstrap/tooltip';

E dentro do “@NgModule” em “imports” insira o nome dos componentes.

  imports: [

    CommonModule,

    BrowserModule,

    AppRoutingModule,

    HttpClientModule,

    FormsModule,

    ModalModule.forRoot(),

    BsDropdownModule.forRoot(),

    TooltipModule.forRoot()

  ],

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

**6 -- 64. Botões e Tooltips**

**Utilizando o tooltip em um botão:**

No codigo html dentro da tag “button” digite ”tooltip=”[Descrição]””.

<button class="btn btn-sm btn-success" tooltip="Editar">

<i class="fa fa-edit"></i>

</button>

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

**6 -- 65. Modal**

**Criando um Modal em html:**

No codigo html digite “<ng-template [Nome do Template]> Conteudo </ng-template>”.

<ng-template #template>

  <div class="modal-header">

    <h4 class="modal-title pull-left">Modal</h4>

    <button type="button" class="close pull-right"

    aria-label="Close" (click)="modalRef.hide()">

    <span aria-hidden="true">&times;</span>

  </button>

</div>

<div class="modal-body">

  This is a modal.

</div>

</ng-template>

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

**Criar uma variavel do tipo Modal em Typescript:**

Importe o conteudo da classe “BsModalRef”.

import { BsModalRef } from 'ngx-bootstrap/modal';

Na classe em Typescript digite “[Nome da variavel]: BsModalRef”.

modalRef: BsModalRef;

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

**Criar um metodo em Typescript para exibir um Modal:**

No construtor receba o “BsModalService”.

constructor(

    private eventoService: EventoService,

    private modalService: BsModalService

    ){}

Dentro do metodo receba o nome do template por parametro e execute o metodo “Show” do “BsModalService” passando o template como parametro.

  openModal(template: TemplateRef<any>){

    this.modalRef = this.modalService.show(template);

  }

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

**Exibir um Modal atraves do Click de um botão:**

Dentro da tag do botão digite “(click)=”[Nome do Metodo]([Parametros])””.

        <button class="btn btn-sm btn-success"

          tooltip="Editar" (click)="openModal(template)">

Obs: Caso esteja utilizando o nome do Modal como parametro nao digitar o #.

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