**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 vários 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 só 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 controller 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 #.

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

**7 -- 68. Criando Formulário**

**Utilizando form-row dentro de uma div:**

Sempre que utilizar uma “div” com “form-row” é necessario informar o tamanho da “div” de dentro.

  <div class="form-row">

    <div class="form-group col-md-12">

      <label>Tema</label>

      <input type="text" class="form-control">

    </div>

  </div>

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

**Como utilizar o Reactive Forms**

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

import { FormsModule, ReactiveFormsModule } from '@angular/forms'

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

  imports: [

    CommonModule,

    BrowserModule,

    AppRoutingModule,

    HttpClientModule,

    FormsModule,

    ModalModule.forRoot(),

    BsDropdownModule.forRoot(),

    TooltipModule.forRoot(),

    BrowserAnimationsModule,

    ReactiveFormsModule

  ],

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

**Reactive Forms no Typescript**

Dentro da classe crie uma propriedade do tipo “FormGroup”.

registerForm: FormGroup;

Crie um metodo para validar os campos.

  validation(){

    this.registerForm = new FormGroup({

      tema: new FormControl,

      local: new FormControl,

      dataEvento: new FormControl,

      qtdPessoas: new FormControl,

      imagemURL: new FormControl,

      telefone: new FormControl,

      email: new FormControl

    });

  }

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

**Reactive Forms no html**

Dentro do “form” utilize o “[formGroup]=”[Nome do FormGroup]””.

<form [formGroup]="registerForm">

Dentro do “input” utilize o “formControlName=”[Nome da Propriedade]””.

<input type="text" class="form-control" formControlName="tema">

Obs: O nome do FormGroup deve ser o mesmo declarado na classe em Typescript.

Obs: O nome da Propriedade deve ser o mesmo utilizado no modelo, para utilizar o retorno do Reactive para realizar um Post na API.

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

**7 -- 69. Validação do Forms - Parte 01**

**Declarando um Form Control**

É possivel criar um FormControl informando no construtor um valor base e as validações.

* Validando que campo precisa ser preenchido.

local: new FormControl('', Validators.required)

* Validando tamanho minimo e máximo da string.

tema: new FormControl('', [Validators.required, Validators.minLength(4), Validators.maxLength(50)]),

* Validando quantidade maxima.

qtdPessoas: new FormControl('', [Validators.required, Validators.max(120000)]),

* Validando email.

email: new FormControl('', [Validators.required, Validators.email])

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

**Exibindo erro caso campo não seja validado**

Dentro do “input” utilize o “[ngClass]” para preencher a string “is-invalid”.

<input type="text" class="form-control"

[ngClass]="{'is-invalid': registerForm.get('tema').errors && registerForm.get('tema').touched}"

     formControlName="tema" placeholder="Insira o Tema">

Caso queira exibir uma mensagem. Crie uma “div” no “class” informe “invalid-feedback”.

<div class="invalid-feedback">

        Imagem é obrigatorio.

</div>

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

**7 -- 70. Validação do Forms - Parte 02**

**Exibindo erro por tipo de validação**

Crie uma “div” para cada erro e depois utilize o “ngIf” para exibir o conteudo caso tenha o erro desejado.

<div class="invalid-feedback"

      \*ngIf="registerForm.get('tema').hasError('required') && registerForm.get('tema').touched">

Tema é obrigatorio.

</div>

<div class="invalid-feedback"

      \*ngIf="registerForm.get('tema').hasError('minlength') && registerForm.get('tema').touched">

Tema deve ter no minimo 4 Caracteres.

</div>

<div class="invalid-feedback"

      \*ngIf="registerForm.get('tema').hasError('maxlength') && registerForm.get('tema').touched">

Tema deve ter no maximo 50 Caracteres.

</div>

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

**7 -- 72. Usando FormBuilder Service**

**Removendo a dependência do FormGroup.**

No construtor receba o “FormBuilder” como parâmetro.

  constructor(

    private eventoService: EventoService,

    private modalService: BsModalService,

    private fb: FormBuilder,

    ){}

No método de validação remova o “FormGroup” e utilize o “FormBuilder.group”.

Remova os “FormControl” e utilize “[‘Valor Default’, [Validators]]”.

  validation(){

    this.registerForm = this.fb.group({

      tema: ['', [Validators.required, Validators.minLength(4), Validators.maxLength(50)]],

      local: ['', Validators.required],

      dataEvento: ['', Validators.required],

      qtdPessoas: ['', [Validators.required, Validators.max(120000)]],

      imagemURL: ['', Validators.required],

      telefone: ['', Validators.required],

      email: ['', [Validators.required, Validators.email]]

    });

  }

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

**Exibir o objeto de validação**

Dentro do html digite o nome da propriedade do tipo “FormGroup” e selecione “value”.

<p>Form Value: {{registerForm.value | json}}</p>

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

**Exibir o status da validação**

Dentro do html digite o nome da propriedade do tipo “FormGroup” e selecione “status”.

<p>Form Status: {{registerForm.status | json}} </p>

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

**Utilizando status do Reactive Form para habilitar um botão**

Dentro do botão insira a propriedade “disabled” verificando se o “FormGroup” esta valido.

<button class="btn btn-primary"

[disabled]="!registerForm.valid" (click)="salvarAlteracao()">

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

**7 -- 73. Date Picker - PT-BR**

**Importando CSS do Date Picker do NGX Bootstrap.**

Dentro da pasta “src” dentro do arquivo “styles.css” importe o conteúdo da classe “datepicker”.

@import '../node\_modules/ngx-bootstrap/datepicker/bs-datepicker.css';

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

**Utilizando o Date Picker do NGX Bootstrap.**

Dentro da pasta “app” dentro do arquivo “app.module.ts” importe o conteúdo da classe “BsDatepickerModule”.

import { BsDatepickerModule } from "ngx-bootstrap/datepicker";

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

  imports: [

    CommonModule,

    BrowserModule,

    AppRoutingModule,

    HttpClientModule,

    FormsModule,

    ModalModule.forRoot(),

    BsDropdownModule.forRoot(),

    BsDatepickerModule.forRoot(),

    TooltipModule.forRoot(),

    BrowserAnimationsModule,

    ReactiveFormsModule

  ],

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

**Utilizando o Date Picker do NGX Bootstrap no html.**

Dentro do input digite “bsDatepicker” .

<input type="text" class="form-control" bsDatepicker

      [ngClass]="{'is-invalid': registerForm.get('dataEvento').errors && registerForm.get('dataEvento').touched}"

      formControlName="dataEvento" placeholder="Insira a Data do Evento">

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

**Alterar o Date Picker do NGX Bootstrap para pt-br.**

Dentro do arquivo em Typescript do componente importe “BsLocaleService”, “defineLocale” e “ptBrLocale”.

import { BsLocaleService } from "ngx-bootstrap/datepicker";

import { defineLocale } from 'ngx-bootstrap/chronos';

import { ptBrLocale } from 'ngx-bootstrap/locale';

Abaixo dos “import” digite:

defineLocale('pt-br', ptBrLocale)

No construtor receba o “BsLocaleService” como parâmetro.

Execute o método “use” do “BsLocaleService” passando como parâmetro “pt-br”.

  constructor(

    private eventoService: EventoService,

    private modalService: BsModalService,

    private fb: FormBuilder,

    private localeService: BsLocaleService

    ){

      this.localeService.use('pt-br');

    }

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

**Alterar o Date Picker do NGX Bootstrap para data e hora.**

Dentro do arquivo em HTML do componente digite “[bsConfig]="{ dateInputFormat: [Formato da Data] ' }"”.

<input type="text" class="form-control"

      bsDatepicker

      [bsConfig]="{ dateInputFormat: 'DD/MM/YYYY hh:mm a' }"

      [ngClass]="{'is-invalid': registerForm.get('dataEvento').errors && registerForm.get('dataEvento').touched}"

      formControlName="dataEvento" placeholder="Insira a Data do Evento">

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

**7 -- 74. Grande MODAL - Organizando Form**

**Exibir qualquer tipo de conteúdo.**

Dentro do método para exibir o Modal defina o parâmetro como any. E execute um show no parâmetro.

openModal(template: any){

    template.show(template);

}

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

**Colocar um espaço no html.**

Dentro de um componente digite “&nbsp”.

    <button class="btn btn-outline-primary" (click)="openModal(template)">

      <i class="fa fa-plus-circle"></i>

      &nbsp;Novo Evento

    </button>

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

**Utilizando o d-flex no html.**

Caso exista mais de uma componente dentro de uma “div” você pode definir o espaço que ele vai ocupar. No parâmetro “class” informe “d-flex”.

* **mr**

No objeto defina o parâmetro “class” como “mr-auto”, para informar que o objeto vai ocupar todo o espaço da esquerda ate à direita. Forcando o outro objeto a ficar colado na direita.

<div class="d-flex">

  <div class="form-inline mr-auto">

    <div class="form-group mb-2">

      <label class="mr-2">Filtro:</label> <input type="text"

      class="form-control mr-2"

      placeholder="buscar"

      [(ngModel)]="filtroLista">

    </div>

  </div>

  <div>

    <button class="btn btn-outline-primary" (click)="openModal(template)">

      <i class="fa fa-plus-circle"></i>

      &nbsp;Novo Evento

    </button>

  </div>

</div>

* **ml**

No objeto defina o parâmetro “class” como “ml-auto”, para informar que o objeto vai ocupar todo o espaço da direita ate à esquerda. Forcando o outro objeto a ficar colado na esquerda.

<div class="modal-footer d-flex">

  <button class="btn btn-secondary" (click)="template.hide()">

    Fechar

  </button>

  <button class="btn btn-primary ml-auto" [disabled]="!registerForm.valid" (click)="salvarAlteracao()">

    Salvar Alterações

  </button>

</div>

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

**7 -- 75. Registrando Evento**

**Copiar todos os valores de um objeto para um objeto vazio em Typescript.**

Dentro do método digite “objeto = Object.assign({}, objeto);”

this.evento = Object.assign({}, this.registerForm.value);

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

**Realizar um Post em Typescript.**

Dentro do método digite “this.http.post([URL],[OBJETO])”;

postEvento(evento: Evento) {

return this.http.post(this.baseURL, evento);

}

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

**Reset no FormGroup.**

Sempre que iniciar um “Template” é importante executar o “reset” no   
“FormGroup”, para limpar o valor de todos os campos e as validações;

openModal(template: any){

this.registerForm.reset();

      template.show(template);

}

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

**Criar um método em Typescript para salvar um objeto.**

Sempre verificar se o “FormGroup” é valido. Depois copiar os dados do objeto e realizar a chamada “http” para salvar os dados.

   salvarAlteracao(template: any){

      if(this.registerForm.valid)

      {

        this.evento = Object.assign({}, this.registerForm.value);

        this.eventoService.postEvento(this.evento).subscribe(

          (novoEvento: Evento) =>{

            console.log(novoEvento);

            template.hide();

            this.getEventos();

          },

          error =>{

            console.log(error);

          }

        );

      }

    }

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

**7 -- 78. Resolvendo Desafio**

**Atualizar os dados em um Reactive Form.**

Execute o método “patchValue([Objeto])”, para atualizar todos os campos informados na criação do “FormGroup”.

    editarEvento(template: any, evento: Evento){

      this.openModal(template);

      this.evento = evento;

      this.registerForm.patchValue(evento);

    }

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

**Copiar todos os valores de um objeto para outro objeto em Typescript.**

Dentro do método digite “objeto = Object.assign({[Campo]: [Valor]}, objeto);”

this.evento = Object.assign({id: this.evento.id}, this.registerForm.value);

Obs: Dessa maneira mantem o id no objeto.

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

**8 -- 82. Criando DTOs**

**Criar um DTO:**

1. Crie a pasta “Dto” caso não exista.
2. Clique com o botão direito do mouse na pasta “Dto” e selecione “New C# Class”.
3. Informe o nome.

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

**8 -- 83. Referências AutoMapper**

**Instalando a referencia ao AutoMapper:**

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

1. Digite “>Nuget Package Manager: Add Package” e pressione enter.
2. Digite “AutoMapper.Extensions.Microsoft.DependencyInjection” e pressione enter.
3. Selecione “AutoMapper.Extensions.Microsoft.DependencyInjection” 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 será pedido para selecionar o projeto que você quer instalar.

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

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

**Criando o AutoMapper**

1. Crie a pasta “Helpers” caso não exista.
2. Clique com o botão direito do mouse na pasta “Helpers” e selecione “New C# Class”.
3. Informe o nome.
4. Importe o comportamento da classe “Profile”.

public class AutoMapperProfiles : Profile

1. No construtor utilize o método “CreateMap<[Objeto], [ObjetoDTO]>”. Para cada objeto que você quer mapear.

public AutoMapperProfiles()

{

CreateMap<Evento, EventoDto>();

   CreateMap<Palestrante, PalestranteDto>();

   CreateMap<Lote, LoteDto>();

   CreateMap<RedeSocial, RedeSocialDto>();

}

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

**Utilizando o AutoMapper**

Dentro da classe “Startup.cs” dentro do método “ConfigureServices” adicione o seguinte conteúdo:

services.AddAutoMapper();

Obs: “AddAutoMapper()” deve estar acima de “AddMvc()”.

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

**8 -- 84. Config Muitos para Muitos**

**Definir como obter o valor da propriedade no AutoMapper:**

Após a execução do método “CreateMap<[Objeto], [DTO]>()” digite “.ForMember()”.

1. Defina no primeiro parâmetro a propriedade do “Objeto”.
2. Através do método “MapFrom()” selecione o valor da propriedade.

CreateMap<Evento, EventoDto>()

.ForMember(dto => dto.Palestrantes, opt =>{

      opt.MapFrom(e => e.PalestranteEventos.Select(x => x.Palestrante)

.ToList());

      });

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

**8 -- 85. DTO na Controller**

**Utilizando o AutoMapper na Controller:**

No construtor da “Controller” receba a interface ”IMapper” como parâmetro.

public EventoController(IProAgilRepository iProAgilRepository, IMapper iMapper)

{

this.IMapper            = iMapper;

      this.IProAgilRepository = iProAgilRepository;

}

Obs: Para receber a interface por meio de injeção, é necessário dentro da classe “Startup.cs” dentro do método “ConfigureServices” adicione o seguinte conteúdo:

services.AddAutoMapper();

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

**Retornando um DTO através de um método da Controller:**

Execute o método “Map<[DTO]>(Objeto)” através da interface “IMapper”.

var eventos = await this.IProAgilRepository.GetAllEventoAsync(true);

var result = this.IMapper.Map<IEnumerable<EventoDto>>(eventos);

return Ok(result);

Obs: Em caso de erro no mapeamento será executado uma exceção informando que mapeamento não foi encontrado.

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

**Criando o mapeamento reverso**

No construtor da classe que implementa o “Profile” utilize uma das duas opções.

* Execute o método “CreateMap<[DTO], [Objeto]>();”

CreateMap<Lote, LoteDto>();

CreateMap<LoteDto, Lote>();

* Utilize o método “ReverseMap()”

CreateMap<Lote, LoteDto>().ReverseMap();

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

**8 -- 86. Controller Post - Dto Reverso**

**Recebendo um DTO através do Post:**

Receba o “DTO” como parâmetro e utilize o método “Map<Objeto>(DTO)” através da interface “IMapper” para realizar o mapeamento reverso.

public async Task<IActionResult> Post(EventoDto model)

{

try

      {

       var evento = this.IMapper.Map<Evento>(model);

            this.IProAgilRepository.Add(evento);

            if (await this.IProAgilRepository.SaveChangesAsync())

            {

            return Created($"/api/evento/{model.Id}", this.IMapper.Map<EventoDto>(evento));

            }

            return BadRequest();

}

      catch (Exception ex)

      {

return this.StatusCode(StatusCodes.Status500InternalServerError, $"Banco de dados Falhou {ex.Message}");

}

}

Obs: Ao tentar utilizar o “DTO” como parâmetro para o método “Add” da interface “IProAgilRepository” será executado uma exceção informando que o objeto não foi encontrado.

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

**8 -- 87. DTO Método PUT**

**Atualizar as propriedades de um objeto através do IMapper:**

Utilize o método “Map([DTO], [OBJETO])” do “Imapper”. Nesse caso as propriedades do “Objeto” serram atualizados pelos dados do “DTO”.

this.IMapper.Map(model, evento);

this.IProAgilRepository.Update(evento);

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

**8 -- 88. Data Annotations**

**Criando validações sobre o objeto recebido:**

Caso você queira criar validações sobre as propriedades dos objetos recebidos pela API, dentro do objeto você pode escrever acima de alguma propriedade uma “Data Annotation”.

* **EmailAddress**: Para validar o e-mail.

[EmailAddress]

public string Email { get; set; }

* **Required**: Definir que campo é de preenchimento obrigatório.

[Required]

public string Tema { get; set; }

* **Phone**: Para validar um telefone.

[Phone]

public string Telefone { get; set; }

* **Range**: Definir que o valor do campo deve estar entre os valores informados.

[Range(2, 120000)]

public int QtdPessoas { get; set; }

* **StringLength**: Definir o tamanho da string.

[StringLength(100, MinimumLength=3)]

public string Local { get; set; }

Obs: Pode existir mais de uma validação por propriedade e também pode ser especificada uma mensagem de erro para cada.

[Required(ErrorMessage="O campo {0} é obrigatorio")] [StringLength(100, MinimumLength=3, ErrorMessage="O campo {0} deve ter entre 3 e 100 caracteres.")]

public string Nome { get; set; }

Obs: O “{0}” vai retornar o nome do campo.

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

**8 -- 89. ApiController**

**Entendendo a utilização da ApiController.**

Se a “Controller” utilizar a “[ApiController]” as validações vão ser executadas automaticamente, caso não utilize a “[ApiController]” é necessário verificar as validações.

if(ModelState.IsValid)

{

return this.StatusCode(StatusCodes.Status400BadRequest, ModelState);

}

Se a “Controller” utilizar a “[ApiController]” os json são obtidos através do corpo, caso não utilize a “[ApiController]” é necessário informar um “Decorator” de parâmetro que informa que o json esta vindo através do corpo.

public async Task<IActionResult> Post([FromBody]EventoDto model)

{

}

Obs: Caso não informe vai tentar obter o json através de “QueryString” da url.

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

**9 -- 91. Adicionando NGX Toastr**

**NGX Toastr:**

Site para download: <https://www.npmjs.com/package/ngx-toastr>

Comando para instalar no terminal: “npm install ngx-toastr --save”.

Obs: Navegue ate a pasta do projeto em Angular com o comando cd.. ou cd [Nome da pasta] antes de instalar.

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

**Habilitar o NGX Toastr para uso:**

Dentro da pasta “app” dentro do arquivo “app.module.ts” importe o conteúdo do NGX Toastr.

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { ToastrModule } from 'ngx-toastr';

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

  imports: [

    CommonModule,

    BrowserModule,

    AppRoutingModule,

    HttpClientModule,

    FormsModule,

    ModalModule.forRoot(),

    BsDropdownModule.forRoot(),

    BsDatepickerModule.forRoot(),

    TooltipModule.forRoot(),

    ToastrModule.forRoot(),

    BrowserAnimationsModule,

    ReactiveFormsModule

  ],

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

**Declarando o CSS**

Dentro do arquivo “angular.json” localize a tag “styles” e insira o conteúdo:

"styles": [

"src/styles.css",

      "node\_modules/ngx-toastr/toastr.css"

],

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

**Utilizando o NGX Toastr no Typescript**

Importe o “ToastrService” na classe em Typescript.

import { ToastrService } from 'ngx-toastr';

No construtor receba o “ToastrService” como parâmetro.

  constructor(

    private eventoService: EventoService,

    private fb: FormBuilder,

    private localeService: BsLocaleService,

    private toastr: ToastrService

    ){

      this.localeService.use('pt-br');

    }

Para exibir uma mensagem de sucesso utilize o método “success([Informação], [Titulo])” do “ToastrService”.

this.toastr.success('Hello world!', 'Toastr fun!');

Para exibir uma mensagem de erro utilize o método “error([Informação], [Titulo])” do “ToastrService”.

this.toastr.error('Hello world!', 'Toastr fun!');

Obs: Caso a mensagem fique transparente no canto superior apenas aparecendo quando o mouse esta sobre ela. Dentro do arquivo “styles.css” adicione o seguinte conteúdo.

#toast-container > div {

    opacity: 1;

}

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

**Definir uma configuração para o Toastr para o projeto todo**

Dentro da pasta “app” dentro do arquivo “app.module.ts” dentro do “@NgModule” em “imports” insira “ToastrModule.forRoot({Configurações})”.

    ToastrModule.forRoot({

      timeOut: 10000,

    positionClass: 'toast-bottom-right',

    preventDuplicates: true,

    }),

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

**9 -- 92. Rápidas Melhorias**

**Definir um Pipe no “value”:**

Dentro do “input” digite.

value="{{dataEvento | DateTimeFormatPipe}}"

[(ngModel)]="dataEvento"

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

**9 -- 94. Adicionando Rotas - Parte 01**

**Criar rotas:**

Ao criar um projeto em “Angular” será perguntado se você deseja criar a parte de rotas. Ao clicar em sim o arquivo “app-routing.module.ts” é criado.

Para criar uma rota abra o arquivo “app-routing.module.ts” e dentro de “Routes” digite “{path:[Caminho utilizado na URL], component: [Nome do Componente]}”.

const routes: Routes = [

  {path: 'eventos', component: EventosComponent},

  {path: 'palestrantes', component: PalestrantesComponent},

  {path: 'dashboard', component: DashboardComponent},

  {path: 'contatos', component: ContatosComponent}

];

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

**Exibindo conteúdo da rotas:**

No código html digite “<router-outlet></router-outlet>” o conteúdo das rotas será exibido nessa tag.

<div class="container">

  <h1>

    Welcome to {{title}}

    </h1>

    <hr>

    <router-outlet></router-outlet>

</div>

Obs: Para ver uma rota basta digitar no navegador “endereço do site/[path de uma rota]”.

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

**9 -- 95. Adicionando Rotas - Parte 02**

**Fazer um link abrir uma rota:**

No código html utilize o “routerLink” passando o “path” da rota como parâmetro.

<li class="nav-item" routerLinkActive="active">

<a class="nav-link" routerLink="contatos" tabindex="-1" aria-disabled="true">Contatos</a>

</li>

Obs: Cuidado ao digitar o “path” porque é Case Sensitive. Tem que ser digitado exatamente igual ao declarado no arquivo “app-routing.module.ts”.

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

**Fazer com que o link exiba qual deles está ativo:**

No código html utilize o “routerLinkActive=”active””.

<li class="nav-item" routerLinkActive="active">

<a class="nav-link" routerLink="contatos" tabindex="-1" aria-disabled="true">Contatos</a>

</li>

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

**9 -- 96. Adicionando Rotas - Parte 03**

**Criar uma rota para caminho vazio:**

Para criar uma rota abra o arquivo “app-routing.module.ts” e dentro de “Routes” digite “{path: '', redirectTo: 'dashboard', pathMatch:'full'}”.

{path: '', redirectTo: 'dashboard', pathMatch:'full'},

Obs: Caso no navegador não seja informada uma rota automaticamente será redirecionado para a rota informada.

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

**Criar uma rota para caminho invalido:**

Para criar uma rota abra o arquivo “app-routing.module.ts” e dentro de “Routes” digite ” {path: '\*\*', redirectTo: 'dashboard', pathMatch:'full'}”.

{path: '\*\*', redirectTo: 'dashboard', pathMatch:'full'}

Obs: Caso no navegador seja informada uma rota invalida, automaticamente será redirecionado para a rota informada.

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

**9 -- 97. Componente Título**

**Criar um Component que será utilizado em vários Components:**

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

Obs: Necessário referenciar o Component na classe “app.module.ts”.

Obs: Não pode utilizar caracteres especiais no nome do “Component”.

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

**9 -- 98. @Input**

**Receber um objeto no Component:**

Na classe em Typescript importe o conteúdo da classe “Input”.

import { Component, Input, OnInit } from '@angular/core';

Depois declare uma variável com “@input” na frente.

@Input() title: string;

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

**Passando um objeto como parâmetro para um Component:**

Para passar uma “String” como parâmetro, no código html digite “[Nome da Propriedade]="'[Conteúdo]'"”.

<app-Titulo [title]="'Eventos'"></app-Titulo>

Para passar um objeto como parâmetro, no código html digite “[Nome da Propriedade]="[Nome do Objeto]"”.

<app-Titulo [title]="titulo"></app-Titulo>

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

**10 -- 101. Upload na Controller**

**Obter um arquivo através de um Controller:**

Para obter os arquivos utilize o “Request.Form.Files”.

var file = Request.Form.Files[0];

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

**Criar um diretório para armazenar imagens:**

1. Dentro do projeto Web crie a pasta “Resources” caso não exista.
2. Dentro da pasta “Resources” crie a pasta “Images” caso não exista.

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

**Obter o diretório atual da aplicação:**

Para obter o diretório atual da aplicação utilize o “Directory.GetCurrentDirectory()”.

string pathToSave = Path.Combine(Directory.GetCurrentDirectory(), folderName);

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

**Obter o nome do arquivo recebido na Controller:**

Para obter o nome do arquivo utilize o “ContentDispositionHeaderValue.Parse([ContentDisposition]).FileName”.

string fileName = ContentDispositionHeaderValue.Parse(file.ContentDisposition).FileName;

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

**Salvar o arquivo recebido na Controller:**

Para salvar um arquivo utilize o “file.CopyTo([Stream]);”

using(var stream = new FileStream(fullPath, FileMode.Create))

{

file.CopyTo(stream);

}

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

**Definir o diretório para obter e salvar arquivos para toda a API:**

Dentro da classe “Startup”, dentro do método “ConfigureServices” insira o seguinte conteúdo. Informando o diretório.

app.UseStaticFiles(new StaticFileOptions(){

FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), @"Resources")),

      RequestPath = new PathString("/Resources")

});

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

**10 -- 102. Botão enviar Arquivo**

**Selecionar arquivo através do html:**

Dentro do “input” defina o “type” para “file”.

<input type="file" #file class="form-control"

    [ngClass]="{'is-invalid': registerForm.get('imagemURL').errors && registerForm.get('imagemURL').touched}"

    formControlName="imagemURL" placeholder="">

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

**Fazer um botão realizar um click no input:**

Crie um identificador para o “input” com “#[Identificador]”.

<input type="file" #file class="form-control"

    [ngClass]="{'is-invalid': registerForm.get('imagemURL').errors && registerForm.get('imagemURL').touched}"

    formControlName="imagemURL" placeholder="">

Utilize a função de “click” do botão para realizar a função de click no “input” através do identificador.

<button class="btn btn-success" (click)="file.click()"></button>

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

**Executar um método em Typescript quando o input alterar:**

Utilize a função de “change” passando um método como parâmetro.

<input type="file" #file class="form-control"

    [ngClass]="{'is-invalid': registerForm.get('imagemURL').errors && registerForm.get('imagemURL').touched}"

    (change)="onFileChange($event)"

    formControlName="imagemURL" placeholder="">

Obs: Ao passar como parâmetro o “$event”, esta passando como parâmetro o próprio evento.

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

**10 -- 103. Evento target File**

**Utilizando uma propriedade do tipo File em Typescript:**

Crie uma propriedade do tipo File.

file: File;

Defina o valor da propriedade do tipo “File”.

if(event.target.files && event.target.files.length)

{

this.file = event.target.files;

}

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

**10 -- 104. Realizando Upload**

**Criar um método em Typescript para realizar o post de uma imagem:**

Crie uma instancia da classe “FormData” e utilize o método “append” para adicionar a imagem.

Depois utilize o método “post” enviando a instancia da classe “FormData” como parâmetro.

postUpload(file: File){

const fileToUpload = <File>file[0];

     const formData = new FormData();

     formData.append('file', fileToUpload, fileToUpload.name);

     return this.http.post(`${this.baseURL}/upload`, formData);

}

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

**Atualizar o nome da imagem no objeto:**

Sempre que obter uma imagem através do “input” o nome do arquivo sera preenchido com “c:\fakepath\[Nome da Imagem]”. Logo antes de salvar o objeto é importante apenas pegar o nome da imagem.

const nomeArquivo = this.evento.imagemURL.split('\\', 3);

this.evento.imagemURL = nomeArquivo[2];

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

**Corrigir erro ao validar imagem:**

Ao utilizar o “Reactive Form” para validar o preenchimento de um “input” com o “type” igual a “file”. Ao executar o método “registerForm.patchValue” para atualizar os dados, você vai receber a mensagem de erro “Failed to set the 'value' property on 'HTMLInputElement'”.

Para resolver o erro basta remover o valor da propriedade que esta sendo validada.

evento.imagemURL = '';

this.registerForm.patchValue(evento);

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

**10 -- 106. Refatoração**

**Criar uma variável dentro de um método em Typescript:**

Para criar uma variável dentro de um método digite “const [Nome] = new [Tipo]”.

const formData = new FormData();

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

**10 -- 108. Forçar Recarregar Grid**

**Forcar a imagem a carregar:**

Caso você queira forcar uma imagem como o mesmo nome a recarregar na tela. Basta passar algum parâmetro para “url” da imagem que altere toda hora.

<img \*ngIf="mostrarImagem"

      src="http://localhost:5000/Resources/Images/{{evento.imagemURL}}?\_ts={{dataAtual}}"

[style.width.px]="imagemLargura"

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

Obs: Nesse caso a propriedade “dataAtual” é preenchida com a data de alteração e passada na “url” como parâmetro. Assim garantindo que a imagem recarregue sempre.

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

**11 -- 111. Criando Entidades**

**Instalando a referencia ao Identity:**

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

1. Digite “>Nuget Package Manager: Add Package” e pressione enter.
2. Digite “Microsoft.AspNetCore.Identity.EntityFrameworkCore” e pressione enter.
3. Selecione “Microsoft.AspNetCore.Identity.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 será pedido para selecionar o projeto que você quer instalar.

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

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

**Criar usuário para Login:**

Crie a classe de usuário implementando o “IdentityUser”.

public class User : IdentityUser<int>

{

public List<UserRole> UserRoles { get; set; }

}

Obs: Caso não seja definido o tipo de ID, será utilizada uma “string” com um “hash”.

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

**Definindo o tipo do dado no banco:**

Acima da propriedade digite “[Column(TypeName="[Tipo do Dado]")]”.

[Column(TypeName="nvarchar(150)")]

public string FullName { get; set; }

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

**Criar regras para usuário:**

Crie a classe de regra implementando o “IdentityRole”.

public class Role : IdentityRole<int>

{

public List<UserRole> UserRoles { get; set; }

}

Obs: Caso não seja definido o tipo de ID, será utilizada uma “string” com um “hash”.

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

**Criar o vinculo entre regra e usuário:**

Crie a classe de vinculo implementando o “IdentityUserRole” e definindo o tipo da variável ID.

public class UserRole : IdentityUserRole<int>

{

public User User { get; set; }

      public Role Role { get; set; }

}

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

**11 -- 112. Criando Relacionamentos**

**Alterar o DBContext:**

Na classe que implementa o “DBContext” altere a implementação para “IdentityDbContext”. Definindo os seguintes tipos na implementação.

public class ProAgilContext : IdentityDbContext<User,

Role,

int,

IdentityUserClaim<int>,

UserRole,

      IdentityUserLogin<int>,

      IdentityRoleClaim<int>,

      IdentityUserToken<int>>

Alterar o método “OnModelCreating” para chamar o método “OnModelCreating” do “IdentityDbContext” passando o parâmetro “ModelBuilder”.

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

}

Alterar o método “OnModelCreating” declarando de forma explicita qual é a “Key” e as “ForeignKey” do objeto.

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<UserRole>(userRole =>{

                userRole.HasKey(ur => new {ur.UserId, ur.RoleId});

                userRole.HasOne(ur => ur.Role)

                    .WithMany(r => r.UserRoles)

                    .HasForeignKey(ur => ur.RoleId)

                    .IsRequired();

                userRole.HasOne(ur => ur.User)

                    .WithMany(r => r.UserRoles)

                    .HasForeignKey(ur => ur.UserId)

                    .IsRequired();

            });

            modelBuilder.Entity<PalestranteEvento>()

            .HasKey(p => new { p.EventoId, p.PalestranteId });

}

Obs: Necessário declarar de forma explicita para evitar erro ao criar a tabela.

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

**Definir uma ForeignKey:**

1. Definir que o “UserRole” tem um “User” através da propriedade “User”.
2. Definir que o “User” tem vários “UserRole” através da propriedade “UserRoles”.
3. Definir que o “UserRole” tem a “ForeignKey” “UserId”.
4. Definir que é de preenchimento obrigatório.

userRole.HasOne(ur => ur.User)

.WithMany(r => r.UserRoles)

      .HasForeignKey(ur => ur.UserId)

      .IsRequired();

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

**11 -- 114. Configurações na Startup**

**Definir a configuração de senha da aplicação:**

Dentro da classe “Startup”, dentro do método “ConfigureServices” insira o seguinte conteúdo.

IdentityBuilder builder = services.AddIdentityCore<User>(options => {

options.Password.RequireDigit = false;

      options.Password.RequireNonAlphanumeric = false;

      options.Password.RequireLowercase = false;

      options.Password.RequireUppercase = false;

      options.Password.RequiredLength = 4;

});

Nesse caso a senha deve ter quatro dígitos e não é obrigada a ter:

1. Um dígito.
2. Um caractere alpha numérico.
3. Um caractere minúsculo.
4. Um caractere maiúsculo.

Obs: Todas as propriedades de “Require” são “true” por padrão.

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

**Definir as regras e o contexto para a aplicação:**

Dentro da classe “Startup”, dentro do método “ConfigureServices” insira o seguinte conteúdo.

builder = new IdentityBuilder(builder.UserType, typeof(Role), builder.Services);

builder.AddEntityFrameworkStores<ProAgilContext>();

builder.AddRoleValidator<RoleValidator<Role>>();

builder.AddRoleManager<RoleManager<Role>>();

builder.AddSignInManager<SignInManager<User>>();

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

**Definir que todas as Controllers precisem de um usuário autenticado.**

Dentro da classe “Startup”, dentro do método “ConfigureServices” insira o seguinte conteúdo.

services.AddMvc(options =>{

var policy = new AuthorizationPolicyBuilder()

                    .RequireAuthenticatedUser()

                    .Build();

options.Filters.Add(new AuthorizeFilter(policy));

})

.SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

Obs: Todo conteúdo adicionado dentro do método “AddMvc” será utilizado por todas as Controllers.

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

**Definir controle de redundância na serialização dos itens.**

Dentro da classe “Startup”, dentro do método “ConfigureServices” execute o método “AddJsonOptions”.

services.AddMvc(options =>{

var policy = new AuthorizationPolicyBuilder()

                    .RequireAuthenticatedUser()

                    .Build();

      options.Filters.Add(new AuthorizeFilter(policy));

})

.SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1)

.AddJsonOptions(opt => opt.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore);

Obs: Controla a redundância na serialização dos itens. Ou seja, caso o objeto A tenha referencia do Objeto B e o objeto B tenha referencia do objeto A, ao serializar o objeto A o sistema não fica em looping infinito.

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

**11 -- 115. JWT**

**Site para explicação sobre o JWT:**

<https://jwt.io/>

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

**Estrutura do Token:**

O Token do JWT é composto por:

1. Informações do Header.
2. Informações do Payload.
3. Informações do Signature.

Obs: Através do exemplo de Token “xxxxxxx.yyyyyyy.zzzzzzz” é possível identificar que cada informação do Token é separada por ponto.

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

**11 -- 116. Configurando JWT**

**Definir a utilização do Token JWT para autenticação na aplicação:**

Dentro da classe “Startup”, dentro do método “ConfigureServices” insira o seguinte conteúdo.

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options=> {

options.TokenValidationParameters = new TokenValidationParameters

       {

       ValidateIssuerSigningKey = true,

            IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(Configuration.GetSection("AppSettings:Token").Value)),

            ValidateIssuer = false,

            ValidateAudience = false,

};

});

Obs: Definição deste arquivo de validação:

1. Definindo que será utilizada a validação da chave do emissor.
2. Definindo tipo de criptografia e a palavra chave utilizada na criptografia.
3. Definindo que não será utilizada validação de “Issuer”.
4. Definindo que não será utilizada validação de “Audience”.

Obs: Nesse caso a palavra chave da criptografia será obtido através do arquivo de configuração.

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

**Criando palavra chave para criptografia:**

Dentro dos arquivos de configuração, “appsettings.json” e “appsettings.Development.json” insira o seguinte conteúdo.

"AppSettings" : {

"Token" : "Super Secret Key"

},

Obs: No Token deve ser digitado a palavra utilizada na criptografia.

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

**11 -- 117. Controller User**

**Criar a Controller de Usuário**

1. Dentro do projeto “WebApi” crie a pasta “Controllers” caso não exista.
2. Clique com o botão direito do mouse na pasta “Controllers” e selecione “New C# Class”.
3. Informe o nome.
4. Faça a nova classe implementar o “ControllerBase”.
5. Receba no construtor o “IConfiguration”, “UserManager”, “SignInManager” e o ” IMapper”.

    [Route("api/[controller]")]

    [ApiController]

    public class UserController : ControllerBase

    {

        public IConfiguration Configuration { get; }

        public UserManager<User> UserManager { get; }

        public SignInManager<User> SignInManager { get; }

        public IMapper Mapper { get; }

        public UserController(IConfiguration configuration,

                                UserManager<User> userManager,

                                SignInManager<User> signInManager,

                                IMapper mapper)

        {

            this.Configuration = configuration;

            this.UserManager = userManager;

            this.SignInManager = signInManager;

            this.Mapper = mapper;

        }

    }

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

**11 -- 119. Método Register**

**Criar o método para registrar um usuário na Controller:**

Utilize o método “CreateAsync” da interface “UserManager” para registrar um usuário.

Para verificar se o usuário foi registrado com sucesso utilize a propriedade “Succeeded” do resultado do método “CreateAsync”.

var result = await this.UserManager.CreateAsync(user, userDto.Password);

if(result.Succeeded)

{

var userToReturn = this.Mapper.Map<UserDto>(user);

      return Created("GetUser", userToReturn);

}

else

{

return BadRequest(result.Errors);

}

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

**Criar um método na Controller que não necessite de autenticação:**

Acima do método digite “[AllowAnonymous]”.

[HttpPost("Register")]

[AllowAnonymous]

public async Task<IActionResult> Register(UserDto userDto)

{

}

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

**11 -- 120. Método Login**

**Criar o método para Login de um usuário na Controller:**

1. Tentar encontrar um usuário registrado através do método “FindByNameAsync()” da interface “UserManager” no banco de dados.
2. Verificar a senha do usuário através do método “CheckPasswordSignInAsync()” da interface “SignInManager”.
3. Caso o resultado da verificação da senha seja positivo recuperar o usuário através do “Users” da interface “UserManager”.

[HttpPost("Login")]

[AllowAnonymous]

public async Task<IActionResult> Login(UserLoginDto userLoginDto)

{

try

      {       var user = await this.UserManager.FindByNameAsync(userLoginDto.UserName);

            var result  = await this.SignInManager.CheckPasswordSignInAsync(user, userLoginDto.Password, false);

            if(result.Succeeded)

            {

                  var appUser = await this.UserManager.Users

                        .FirstOrDefaultAsync(u => u.NormalizedUserName == userLoginDto.UserName.ToUpper());

                  var userToReturn = this.Mapper.Map<UserLoginDto>(appUser);

                  return Ok(new {

                        token = GenerateJWToken(appUser).Result,

                        user = userToReturn

                  });

            }

            else

            {

             return Unauthorized();

            }

}

      catch(Exception ex)

      {

            return this.StatusCode(StatusCodes.Status500InternalServerError, $"Banco de dados Falhou {ex.Message}");

}

}

Obs: Caso o usuário seja encontrado ira retornar um objeto com o Usuário e o Token.

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

**11 -- 121. Gerando o TOKEN**

**Criar método para gerar o Token:**

Criar a lista de “Claim” com os dados do usuário.

var claims  = new List<Claim>

{

new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),

      new Claim(ClaimTypes.Name, user.UserName)

};

Recuperar as “Role” do usuário através do método “GetRolesAsync()” da interface “UserManager”. E adicionar a lista de “Claim” cada “Role” do usuário.

var roles = await this.UserManager.GetRolesAsync(user);

foreach(var role in roles)

{

claims.Add(new Claim(ClaimTypes.Role, role));

}

Recuperar a chave de criptografia através do arquivo de configuração.

var key = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(this.Configuration.GetSection("AppSettings:Token").Value));

Definir as credenciais de assinatura passando a chave e o tipo de criptografia.

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);

Criar a descrição do Token, definindo o assunto, data de expiração e as credenciais de assinatura.

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(claims),

      Expires = DateTime.Now.AddDays(1),

      SigningCredentials = creds,

};

Gerar o Token através do método “WriteToken” da classe “JwtSecurityTokenHandler”.

var tokenHandler = new JwtSecurityTokenHandler();

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

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

**11 -- 122. Registrando Usuário**

**Debugar projeto em execução:**

Para debugar um projeto em execução:

1. Clique na aba lateral sobre o Run “Icone de um inseto” ou pelo atalho Ctrl + Shift + D.
2. Depois clique sobre a seta para baixo nas opções de “Debugging” e selecione “.NET Core Attach”.
3. Clique sobre o ícone da seta verde “Start Debugging”.
4. Será exibida uma janela para selecionar o processo que esta em execução na sua maquina.
5. Selecione o processo anexado ao projeto em WebApi.

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

**11 -- 123. Login do Usuário**

**Erro de usuário não autenticado mesmo informando um Token valido:**

Dentro da classe “Startup”, dentro do método “Configure” insira o seguinte conteúdo.

app.UseAuthentication();

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

**12 -- 125. Criando Login e Registration**

**Criar os componentes de usuário no projeto em Angular:**

1. Clique sobre a pasta “app” com o botão direito do mouse e selecione “Generate Component”.
2. Digite o nome do Component. No caso do usuário “user”.
3. Clique sobre a pasta “user” com o botão direito do mouse e selecione “Generate Component”.
4. Digite o nome do Component. Nesse caso “Login”.
5. Repita os passos três e quatro para criar o Component “Registration”.

Obs: Sempre que criar um Component dentro de outro Component, a extensão do Visual Studio Code que tem o “Generate Component” não vai criar a referencia desses componentes no “app.module.ts”. Sendo necessário importar as referencias desses componentes dentro da “app.module.ts” e informar os nomes dos componentes dentro do “declarations”.

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

**Criar um Component html que carregue outros componentes:**

Dentro do código html do Component utilize o “router-outlet” para poder carregar os Componentes dentro dele.

<div class="container">

  <div class="col-md-8 offset-md-2">

    <router-outlet></router-outlet>

  </div>

</div>

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

**Definir uma rota para um Component com dependências:**

Dentro da classe “app-routing.module.ts” dentro do “Routes” adicione o caminho para o Component depois defina o “children” informando o caminho para cada dependente.

  {path: 'user', component: UserComponent,

  children:[

    { path: 'login', component: LoginComponent },

    { path: 'registration', component: RegistrationComponent }

  ]},

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

**12 -- 126. Inserindo Código Fonte**

**Utilizar o Router no html:**

Para utilizar os métodos da classe “Router” no código html de um componente.

<button class="btn btn-lg btn-link btn-block" (click)="router.navigate(['/user/registration'])">Quero me cadastrar</button>

Na classe em Typescript do componente, no construtor defina a propriedade “Router” como publica.

constructor(public router: Router) { }

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

**12 -- 127. Tela Registrar Usuário**

**Utilizar o Reactive Form para criar um FormGroup dentro de outro FormGroup:**

No código em html do componente crie um ”form” informando o “formGroup”, depois crie uma “div” informando o “formGroupName”.

<form [formGroup]="registerForm">

<div formGroupName="passwords">

No código em Typescript do componente dentro do método de validação e dentro do método “group” da interface “FormBuilder”, informe o nome do “formGroupName” e utilize o método “group” da interface “FormBuilder” para criar um novo grupo de validação.

validation(){

this.registerForm = this.fb.group({

      fullName: [ '', [Validators.required]],

      email: ['', [Validators.required, Validators.email]],

      userName: ['', Validators.required],

      passwords: this.fb.group({

        password: ['', [Validators.required, Validators.minLength(4)]],

        confirmPassword: [ '', [Validators.required, Validators.minLength(4)]]

      })

    });

  }

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

**12 -- 128. Função Comparar Senha**

**Criar método em Typescript para utilizar no Reactive Form:**

Crie um método que receba um “FormGroup” como parâmetro, depois realize as validações que desejar.

compararSenhas(fb: FormGroup){

const confirmSenhaCtrl = fb.get('confirmPassword');

if(confirmSenhaCtrl.errors == null || 'mismatch' in confirmSenhaCtrl.errors)

{

if(fb.get('password').value != confirmSenhaCtrl.value)

      {

       confirmSenhaCtrl.setErrors({ mismatch: true });

      } else {

       confirmSenhaCtrl.setErrors(null);

       }

}

}

Obs: Para recuperar os dados de uma determinada propriedade utilize o método “get” através da classe “FormGroup”.

Obs: Para criar um erro utilize o método “setErrors” da classe “AbstractControl” informando “{ [Nome do Erro]: true }.

Obs: Para verificar se um determinado erro existe digite “[Nome do Erro] in AbstractControl.errors”.

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

**Passando método de validação para o Reactive Form:**

Dentro do método “group” da classe “FormGroup”, após passar o objeto digite “{Objeto}, { validators: [Metodo de Validacao] }”.

this.fb.group({

        password: ['', [Validators.required, Validators.minLength(4)]],

        confirmPassword: [ '', [Validators.required, Validators.minLength(4)]]

      }, { validators: this.compararSenhas })

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

**12 -- 129. Nav e Auth Service**

**Criar Service para controle de login no Angular:**

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 “auth”.

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

**12 -- 130. Auth Service**

**Site para instalar o Jwt no Angular:**

<https://www.npmjs.com/package/@auth0/angular-jwt>

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

**Comando para instalar o Jwt no Angular:**

No terminal digite “npm i @auth0/angular-jwt”.

Obs: Navegue ate a pasta do projeto em Angular com o comando cd.. ou cd [Nome da pasta] antes de instalar.

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

**Utilizando o map em uma classe em Typescript:**

Importe o conteúdo do “map”.

import { map } from 'rxjs/operators';

Utilize o “map” dentro do método “pipe” para tratar o retorno do método “post”.

  login(model: any) {

    return this.http.post(`${this.baseURL}Login`, model).pipe(

      map((response: any) => {

        const user = response;

        if(user)

        {

          localStorage.setItem('token', user.token);

          this.decodedToken = this.jwtHelper.decodeToken(user.token);

        }

      })

    );

  }

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

**Utilizando o localStorage para armazenar um dado:**

O “localStorage” é o armazenamento local do seu browser. Para armazernar um dado utilize o método “setItem” da classe “localStorage”.

localStorage.setItem('token', user.token);

Para obter um dado armazernado utilize o método “getItem” da classe “localStorage”.

 const token = localStorage.getItem('token');

Obs: Poderia ser utilizado o “cookie” no lugar do “localStorage”, mas qualquer pessoa conseguiria recuperar o “token” através do “cookie”. Com o “localStorage” apenas a mesma url consegue recuperar o “token”.

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

**Utilizando o Jwt em uma classe em Typescript:**

Importe o conteúdo do “JwtHelperService”.

import { JwtHelperService } from '@auth0/angular-jwt';

Crie uma propriedade que receba a classe “JwtHelperService”.

jwtHelper = new JwtHelperService();

Utilizar o método “decodeToken()” da classe “JwtHelperService” para decodificar o token.

this.decodedToken = this.jwtHelper.decodeToken(user.token);

Utilizar o método “isTokenExpired ()” da classe “JwtHelperService” para verificar se o token esta expirado.

 return this.jwtHelper.isTokenExpired(token);

Obs: Não é possível descriptografar todos os dados do token apenas o “Header” e o “Payload”.

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

**12 -- 131. Registrar Usuário - Parte 01**

**Obter o valor de uma propriedade de um FormGroup dentro de outro FormGroup:**

Para obter o valor utilize o método “get()” da classe “FormGroup” informando “[Nome do FormGorup].[Nome da Propriedade]”.

      this.user = Object.assign(

        { password: this.registerForm.get('passwords.password').value },

        this.registerForm.value);

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

**12 -- 132. Registrar Usuário - Parte 02**

**Navegar para outro pagina através do Typescript:**

No construtor da classe em Typescript receba a classe “Router” como parâmetro.

constructor(private router: Router) { }

Utilize o método “navigate” da classe “Router” informando a rota da pagina.

 this.authService.register(this.user).subscribe(

          () => {

            this.toastr.success('Cadastro Realizado!')

            this.router.navigate(['/user/login']);

          }

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

**Tratar erro de subscribe no Typescript:**

Utilize a propriedade “code” para realizar uma verificação e criar um retorno por tipo de erro.

this.authService.register(this.user).subscribe(

() => {

this.toastr.success('Cadastro Realizado!')

      this.router.navigate(['/user/login']);

},

error => {

const erro = error.error;

      erro.forEach(element => {

      switch (element.code) {

       case 'DuplicateUserName':

            this.toastr.error('Cadastro Duplicado!');

            break;

            default:

            this.toastr.error(`Erro no Cadastro! CODE: ${element.code}`);

            break;

      }

});

});

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

**12 -- 133. Login**

**Passar para o método Post um Objeto:**

Declare o objeto como qualquer e recebendo um objeto vazio.

model: any = {};

Ou declare o objeto com um tipo especifico e recebendo um novo objeto.

model: User = new User();

Utilize o Two way data binding para preencher o objeto.

<input type="email" id="username" class="form-control" placeholder="Usuário"

    name="username" required [(ngModel)]="model.userName">

Obs: Caso utilize o objeto vazio ele vai criar a propriedade com o valor dentro. Nesse exemplo a propriedade se chamaria “userName” dentro do objeto “model”.

Passar o objeto para o método “post”.

this.authService.login(this.model).subscribe();

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

**Limpar os dados dentro do LocalStorage:**

1. Dentro do site clique com o botão direito do mouse e selecione “Inspecionar”.
2. Na aba principal que contem “Elementos”, “Console” e outros, clique sobre “Aplicativo”.
3. Clique sobre “Armazenamento Local” para exibir todos os sites.
4. Clique com o botão direito do mouse sobre o site e selecione “Apagar”.

Obs: No ultimo passo também é possível clicar sobre o site e apagar apenas uma variável ou visualizar o valor da mesma.

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

**Remover um item do LocalStorage através do Typescript:**

Utilize o método “removeItem()” da classe “localStorage” informando o nome do item.

localStorage.removeItem('token');

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

**Exibir mensagem com fundo preto:**

Utilize o método “show()” da classe “ToastrService” informando a mensagem.

this.toastr.show('Log Out');

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

**Exibir a mão ao posicionar o mouse sobre o componente:**

Dentro do componente digite “style="cursor: pointer;"”.

<a class="dropdown-item" (click)="logout()" style="cursor: pointer;">

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

**Exibir no html o nome do usuário utilizado no Token:**

Utilize a propriedade “unique\_name” do objeto gerado através do método “decodeToken” da classe “JwtHelperService”.

<a dropdownToggle class="nav-link dropdown-toggle" id="navbarDropdownMenuLink" data-toggle="dropdown"

          aria-haspopup="true" aria-expanded="false" style="cursor: pointer;">

          {{authService.decodedToken?.unique\_name}}

        </a>

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

**12 -- 134. Header Autenticado**

**Passar o Token para um método get:**

Crie uma instancia da classe “HttpHeaders” passando um novo objeto como parâmetro com a propriedade “Authorization” e o valor de “Bearer [Seu Token]”.

Passe para o método “get” a url depois um novo objeto com a propriedade “headers” e o valor com a instancia da classe “HttpHeaders”

getAllEvento() : Observable<Evento[]>{

const token = localStorage.getItem('token');

const tokenHeader = new HttpHeaders({ 'Authorization': `Bearer ${token}` });

return this.http.get<Evento[]>(this.baseURL, { headers: tokenHeader});

}

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

**Para visualizar um erro detalhadamente no navegador:**

1. Dentro do site clique com o botão direito do mouse e selecione “Inspecionar”.
2. Na aba principal que contem “Elementos”, “Console” e outros, clique sobre “Rede”.
3. Caso não exiba a lista de Urls pressione “Ctrl” + “R”.
4. Selecione a url com erro.

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

**12 -- 135. Angular Gard**

**Gerando o Gard:**

1. Dentro da pasta “app” crie a pasta “auth” caso não exista.
2. No terminal navegue ate a pasta do projeto em Angular com o comando cd.. ou cd [Nome da pasta].
3. No terminal digite “ng g g auth/auth”.
4. Selecione a opção “CanActivate”.

Obs: Explicando o comando “ng g g auth/auth”:

Primeiro g é de generate.

Segundo g é de gard.

O auth/auth é o diretório e o nome do arquivo.

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

**Configurando o Gard para verificar se o Token é valido:**

Dentro do método “canActivate” verifique se o token é valido.

  canActivate(

    route: ActivatedRouteSnapshot,

    state: RouterStateSnapshot): boolean {

    if(this.authService.loggedIn())

    {

      return true;

    }

    else

    {

      this.router.navigate(['/user/login']);

      return false;

    }

  }

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

**Definindo o Gard para cada rota:**

Abra o arquivo “app-routing.module.ts” e dentro de “Routes” ao definir uma rota informe a propriedade “canActivate” passando a classe “AuthGuard” como parâmetro.

{path: 'eventos', component: EventosComponent, canActivate: [AuthGuard]},

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

**12 -- 136. Interceptor**

**Gerando o Interceptor:**

1. Dentro da pasta “app” crie a pasta “auth” caso não exista.
2. Clique com o botão direto do mouse sobre a pasta “auth” e selecione “Generate Class”.
3. Digite o nome “auth.Interceptor”.

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

**Configurando o Interceptor para alterar cada requisição http que sai da aplicação:**

Importe e utilize o “Injectable” na classe.

import { Injectable } from '@angular/core'

@Injectable({

    providedIn: 'root'

})

Alterar o nome da classe para “AuthInterceptor ” e implemente o “HttpInterceptor ”.

export class AuthInterceptor implements HttpInterceptor {

Implemente o método “intercept” que sera executado toda vez que o aplicativo fizer uma requisição http.

intercept(req: HttpRequest<any>, next: HttpHandler) : Observable<HttpEvent<any>> {

        if(this.authService.loggedIn())

        {

            const token = localStorage.getItem('token');

            const cloneReq = req.clone({

                headers: req.headers.set('Authorization', `Bearer ${token}`)

            });

            return next.handle(cloneReq).pipe(

                tap(

                    succ => {},

                    error => {

                        if(error.status == 401)

                        {

                            this.router.navigateByUrl('user/login');

                        }

                    }

                )

            )

        }

        else

        {

            return next.handle(req.clone());

        }

Obs: É clonada a requisição e adicionado ao “headers” o “Authorization” com o “Token”.

Obs: Depois é utilizado o método “pipe” passando um “tap” para tratar o retorno. O “tap” é utilizado para tratar múltiplas requisições.

Obs: Quando for importar o “tap” utilizar o caminho completo para evitar problema de conflito.

import { tap } from 'rxjs/internal/operators/tap';

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

**Definindo o Interceptor para utilização:**

Abra o arquivo “app.module.ts” e dentro de “providers” crie um novo objeto definindo:

* **Provide:** Com a classe “HTTP\_INTERCEPTORS”.
* **useClass:** Com a classe criada para realizar a interceptação.
* **multi:** Com true para informar que aceita múltiplas requisições.

  providers: [

    EventoService,

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: AuthInterceptor,

      multi: true

    }

  ],

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

**13 -- 139. Criando e Roteando Evento Edit**

**Adicionar um rota com parâmetros:**

Para criar uma rota com parâmetros, abra o arquivo “app-routing.module.ts” e dentro de “Routes” digite no “path” [Nome do Caminho]/:[Parâmetro].

{path: 'evento/:id/edit', component: EventoEditComponent, canActivate: [AuthGuard]},

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

**13 -- 140. Copiando Códigos**

**Utilizar ancora no html:**

Dentro do código html digite “<a></a>”:

<a [routerLink]="['/evento', evento.id, 'edit']" tooltip="Edição Detalhada" class="btn btn-sm btn-warning">

    <i class="fa fa-edit"></i>

</a>

Obs: Para definir a aparência da ancora como um botão digite “class="btn btn-sm btn-warning"”.

Obs: Caso seja clicado sobre a ancora sera executado o “routerLink”, que nesse caso esta criando uma url concatenando os valores informados. Ex: “/evento/1/edit”.

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

**Importando o tabset do NGX Bootstrap:**

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

import { TabsModule } from 'ngx-bootstrap/tabs'

E dentro do “@NgModule” em “imports” insira o “TabsModule ”.

   TabsModule.forRoot(),

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

**Utilizando o tabset no código html:**

No codigo html dentro da “div” digite ”<tabset>”.

<div class="col-md-8">

      <tabset>

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

**Site para instalar o NGX Mask:**

<https://www.npmjs.com/package/ngx-mask>

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

**Instalar o NGX Mask:**

No terminal digite “npm i ngx-mask”.

Obs: Navegue ate a pasta do projeto em Angular com o comando cd.. ou cd [Nome da pasta] antes de instalar.

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

**Importando o NGX Mask:**

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

import { NgxMaskModule } from 'ngx-mask'

E dentro do “@NgModule” em “imports” insira o “NgxMaskModule”.

NgxMaskModule.forRoot(),

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

**Utilizar o NGX Mask no html:**

Dentro do código html digite “{{[Objeto] | mask: ‘[Formatacao]’}}”:

<small class="text-muted">Telefone: {{evento.telefone | mask: '(000) 00000-0000'}}</small>

Ou utilize a propriedade “mask” definindo a formatação.

  <input type="text" mask="(000) 0000-0000" class="form-control" [(ngModel)]="evento.telefone" [ngClass]="{'is-invalid': registerForm.get('telefone').errors

  && registerForm.get('telefone').touched}" formControlName="telefone" placeholder="(000) 90000-0000">

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

**13 -- 143. Trocando a Imagem**

**Anexar imagem a projeto em Angular:**

1. Dentro da pasta “src” crie a pasta “assets” caso não exista.
2. Dentro da pasta “assets” crie a pasta “img” caso não exista.
3. Atraves do “explorer” coloque a imagem dentro da pasta.

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

**Carregar imagem no Typescript:**

Execute o método “readAsDataURL” da classe “FileReader” e implemente o método “onload” para obter a url da imagem.

  onFileChange(file: FileList){

    const reader = new FileReader();

    reader.onload = (event: any) => this.imagemURL = event.target.result;

    reader.readAsDataURL(file[0]);

  }

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

**13 -- 144. Form Array Group**

**Utilizar o array do FormGroup no Typescript:**

Dentro do método que cria o “FormGroup” declare uma propriedade que vai recever um array.

redesSociais: this.fb.array([this.criaRedeSocial()])

Metodo que devolve um novo “FormGroup”.

  criaRedeSocial() : FormGroup {

    return this.fb.group({

      nome: ['', Validators.required],

      url: ['', Validators.required]

    });

  }

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

**Utilizar o formArrayName no html:**

Utilize a propriedade “formArrayName” informando o nome do Form Array Group. E depois utilize o “\*ngFor” para exibir o conteúdo por valor do array.

<div formArrayName="redesSociais" \*ngFor="let redeSocial of this.registerForm.get('redesSociais').value; let i=index">

<fieldset [formGroupName]="i" class="form-group">

Obs: Dentro do “fieldset” a propriedade “formGroupName” esta entre [] porque esta sendo passada uma variável para ela.

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

**13 -- 145. FormArrayGroup - Adicionar e Remover Item**

**Alterar uma interface para uma classe:**

Dentro do objeto altere o tipo de “interface” para “class” e crie um “constructor” vazio.

export class Evento {

    constructor() { }

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

**Utilizando a classe no Typescript:**

Digite “[Nome da Propriedade: Tipo = new Tipo();]”.

evento: Evento = new Evento();

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

**Criar propriedade para obter valor do Form Array Group:**

Digite “get [Nome da propriedade]() : FormArray {}”.

  get lotes() : FormArray {

    return <FormArray>this.registerForm.get('lotes');

  }

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

**Utilizando propriedade do Form Array Group no html:**

Utilize o “\*ngFor” passando a “[Nome da Propriedade]. controls” para ler o conteúdo do Array.

<div formArrayName="lotes" \*ngFor="let lote of lotes.controls; let i=index">

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

**Adicionando objetos ao Form Array Group no Typescript:**

Utilize o método “push” da propriedade “FormArray” passando um “FormGroup”.

adicionarLote() {

    this.lotes.push(this.criaLote());

  }

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

**Remover objetos do Form Array Group no Typescript:**

Utilize o método “removeAt” da propriedade “FormArray” passando um “FormGroup”.

  removerLote(id: number) {

    this.lotes.removeAt(id);

  }

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

**13 -- 146. Carregando Evento**

**Recuperar parâmetro da url:**

Receba no construtor a classe “ActivatedRoute”.

  constructor(private fb: FormBuilder,

              private eventoService: EventoService,

              private activatedRoute: ActivatedRoute) { }

Utilize o método “get” através da classe “ActivatedRoute” para obter o parâmetro.

const idEvento = +this.activatedRoute.snapshot.paramMap.get('id');

Obs: Foi utilizado o “+” para converter a string para um “number”.

Obs: O nome do parâmetro deve ser o mesmo utilizado para criação da rota dentro da classe “app-routing.module.ts”.

{path: 'evento/:id/edit', component: EventoEditComponent, canActivate: [AuthGuard]},

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

**13 -- 147. Carregando Lote e Rede Social**

**Criar FormGroup passando valores default:**

Digite “Nome da Propriedade: [ Valor, Validators ]”.

  criaRedeSocial(redeSocial: any) : FormGroup {

    return this.fb.group({

      id: [redeSocial.id],

      nome: [redeSocial.nome, Validators.required],

      url: [redeSocial.url, Validators.required]

    });

  }

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

**13 -- 148. Validação Dinâmica**

**Obter valor de uma propriedade dentro de um Form Array Group:**

Atraves da propriedade que retorna um “FormArray” utilize o método “get()” passando o index mais “.[Nome da propriedade]”.

Ex: “lotes.get(i+'.nome')”.

<div formArrayName="lotes" \*ngFor="let lote of lotes.controls; let i=index">

<fieldset [formGroupName]="i" class="form-group">

          <div class="form-group col-md-4">

            <label>Nome</label>

            <input type="text" class="form-control" [ngClass]="{'is-invalid': lotes.get(i+'.nome').errors && lotes.get(i+'.nome').touched}" formControlName="nome" placeholder="Lote">

          </div>

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

**13 -- 150. Deletando Relacionamento**

**Deletar um objeto relacionado caso não seja informado no JSON:**

Dentro da api caso seja recebido um objeto através do JSON, é necessário fazer uma verificação nos objetos relacionados para verificar se os objetos continuam sendo informados e removelos caso não.

public async Task<IActionResult> Put(int eventoId, EventoDto model)

{

try

      {

                Evento evento = await this.IProAgilRepository.GetEventoByIdAsync(eventoId);

                if (evento == null)

                {

                    return NotFound();

                }

                evento.Lotes.ForEach(lote =>

                {

                    if(!model.Lotes.Any(l => l.Id == lote.Id))

                    {

                        this.IProAgilRepository.Delete(lote);

                    }

                });

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

**13 -- 153. Mascara R$ (Dinheiro) Reais**

**Site para instalar o NGX Currency:**

<https://www.npmjs.com/package/ngx-currency-mask>

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

**Instalar o NGX Currency:**

No terminal digite “npm i ngx-currency-mask”.

Obs: Navegue ate a pasta do projeto em Angular com o comando cd.. ou cd [Nome da pasta] antes de instalar.

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

**Importando o NGX Currency:**

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

import { CurrencyMaskModule } from "ngx-currency-mask";

E dentro do “@NgModule” em “imports” insira o “CurrencyMaskModule”.

CurrencyMaskModule,

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

**Utilizar o NGX Currency no html:**

Dentro do código html digite “currencyMask ”.

<input currencyMask [options]="{ prefix: 'R$ ', thousands: '.', decimal: ',', align:'left' }"

Obs: Caso queira alguma alteração pode utilizar o “[options]” informando um objeto como parâmetro.

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

**Utilizar o prefix do NGX Mask no html:**

Dentro do código html digite “prefix” e defina o valor inicial do campo.

<input type="text" mask="" prefix="http://"

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

**13 -- 154. Resolvendo Nome Logado**

**Utilizar o sessionStorage para armazenar uma variável.**

Para armazenar uma variável utilize o método “setItem()” da classe “sessionStorage”.

sessionStorage.setItem('userName', this.decodedToken.unique\_name)

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

**Utilizar o sessionStorage para obter o valor de uma variável.**

Para obter um valor armazenado utilize o método “getItem()” da classe “sessionStorage”.

const userName = sessionStorage.getItem('userName');

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

**14 -- 157. Bootswatch**

**Site para instalar o Bootswatch:**

<https://www.npmjs.com/package/bootswatch>

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

**Site para olhar os temas do Bootswatch:**

<https://bootswatch.com/>

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

**Instalar o Bootswatch:**

No terminal digite “npm i bootswatch”.

Obs: Navegue ate a pasta do projeto em Angular com o comando cd.. ou cd [Nome da pasta] antes de instalar.

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

**Utilizar o Bootswatch no projeto:**

Dentro da pasta “src” dentro do arquivo “styles.css” importe o tema do “bootswatch”

@import '../node\_modules/bootswatch/dist/lumen/bootstrap.min.css';

Obs: Nesse caso o tema importado foi o “lumen”.

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

**14 -- 158. Solução do Desafio**

**Obter a url atual através do Typescript:**

Utilize a propriedade “url” da classe “Router”.

return this.router.url != "/user/login";

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

**14 -- 159. Dashborad**

**Site para obter layouts completos:**

<https://bootsnipp.com/>

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

**14 -- 160. Deploy Angular**

**Site para obter os comandos para gerar o Deploy da aplicação:**

<https://angular.io/guide/deployment>

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

**Instalar o Source Map Explorer:**

No terminal digite “npm install source-map-explorer --save-dev”.

Obs: Importante instalar para alterar o nome dos arquivos durante o deploy. Uma vez que caso não mude o nome dos arquivos ao subir no servidor o site pode não carregar os arquivos. Igual ocorre com as imagens.

Obs: Navegue ate a pasta do projeto em Angular com o comando cd.. ou cd [Nome da pasta] antes de instalar.

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

**Gerar o Deploy:**

No terminal digite “ng build --prod --source-map”.

Obs: Ao terminar a execução sera gerado o diretório “dist” dentro do diretório da aplicação.

Obs: Navegue ate a pasta do projeto em Angular com o comando cd.. ou cd [Nome da pasta] antes de instalar.

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

**Executar um projeto em Angular em uma porta especifica:**

No terminal digite. “ng serve --port 4201 -o”.

Obs: “-o” Serve para abrir o browser.

Obs: Você deve estar na pasta do projeto.

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

**14 -- 161. Publish Dotnet Web API**

**Entender o comando de publish:**

No terminal digite “dotnet publish -h”.

Obs: Sera exibido todas as variações do comando publish.

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

**Publicar o aplicativo em um diretório especifico:**

No terminal digite “dotnet publish -o C:\Users\Rafael\Desktop\Teste”.

Obs: Caso diretório não exista sera o mesmo sera criado.

Obs: Necessario criar o diretório “Resources” manualmente dentro da pasta que o projeto foi publicado. Para evitar erro ao executar a aplicação.

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

**Executar o Publish:**

No terminal digite “dotnet ProAgil.WebAPI.dll”.

Obs: Navegue ate a pasta aonde o projeto foi publicado.

Obs: Para saber qual é o nome do arquivo utilize o comando “dir”.

Obs: Em caso de erro no banco de dados copiar o banco da pasta do projeto e substituir na pasta onde o projeto foi publicado.

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

**15 -- 163. Mostrando Docker e Instalando EF Core SQL**

**Instalar extensão do SQL Server (mssql):**

Clique no icone “Extensions” na barra ao lado esquerdo ou pressione Ctrl + Shift + X. Depois digite “SQL Server (mssql)” e clique sobre “install”.

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

**Instalar extensão do Docker:**

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

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

**Adicionando string de conexão do SQL Server:**

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

 "ConnectionStrings": {

    "DefaultConnection": "Data Source=ProAgil.db",

    "DefaultConnectionSQL": "Server=127.0.0.1\\SQLEXPRESS;Database=ProAgil;User Id=sa;Password=habibs;"

  },

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

**Instalando o SQL Server:**

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.SqlServer” e pressione enter.
3. Selecione “Microsoft.EntityFrameworkCore.SqlServer” 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”.

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

**Definindo o SQL Server como conexão para o contexto da aplicação:**

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

services.AddDbContext<ProAgilContext>(d => d.UseSqlServer(Configuration.GetConnectionString("DefaultConnectionSQL")));

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

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

1. É necessario ter instalado o SQL Server antes. É possivel verificar se o SQL Server esta instalado no arquivo “[NomeDoProjeto].csproj” caso existe a linha”:

<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="2.2.6"/>

1. Caso esteja instalado e não apareca apenas execute o comando no terminal “dotnet build”.

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

**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 SQL Server 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 SQL Server no projeto.

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

**Executar um comando no banco de dados SQL Server:**

1. No Visual Studio Code pressione “Ctrl” + “N”.
2. Sera aberto um arquivo com o nome “Untitled”.
3. Digite no arquivo “Untitled” o comando do banco de dados Ex: SELECT \* FROM eventos.
4. Pressione “Ctrl” + “Shift” + “P”.
5. Digite “>MS SQL: Execute Query”.
6. Sera perguntado se deseja alterar a linguagem, selecione “Yes” e pressione o “Enter”.
7. Sera pedido para selecionar a linguagem, selecione “SQL” e pressione o “Enter”.
8. Caso já tenha uma conexão selecione a mesma e pressione o “Enter”, caso contrario selecione “Create Connection Profile” e pressione o “Enter”.
9. Informe o nome do servidor e pressione o “Enter”. Ex: “127.0.0.1\SQLEXPRESS”.
10. Informe o nome do banco de dados e pressione o “Enter”. Ex: “ProAgil”.
11. Selecione “SQL Login” e pressione o “Enter”.
12. Informe o nome do usuário e pressione o “Enter”. Ex: “sa”.
13. Informe a senha e pressione o “Enter”. Ex: “habibs”.
14. Sera perguntado se deseja salvar a configuração, selecione “Yes” e pressione o “Entrer”.
15. Informe o nome dessa conexão e pressione o “Enter”. Ex: “Teste”.

Obs: Caso sua conexão seja realizada com sucesso sera exibida uma janela com o resultado do seu comando no banco de dados.

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

**Lembrar porque o projeto Web APi não precisa referenciar o Entity Framework:**

Ao adicionar as referencias do “Microsoft.AspNetCore.App” e “Microsoft.AspNetCore.Razor.Design” elas já incluem a referencia ao “EntityFramework”.

    <PackageReference Include="Microsoft.AspNetCore.App"/>

    <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.1.2" PrivateAssets="All"/>

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