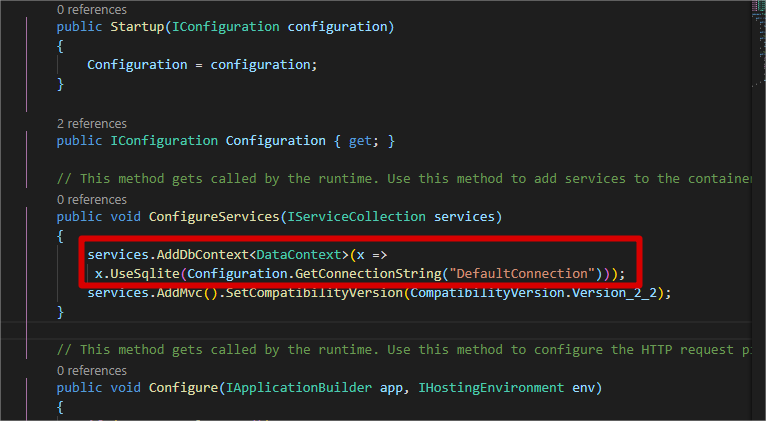
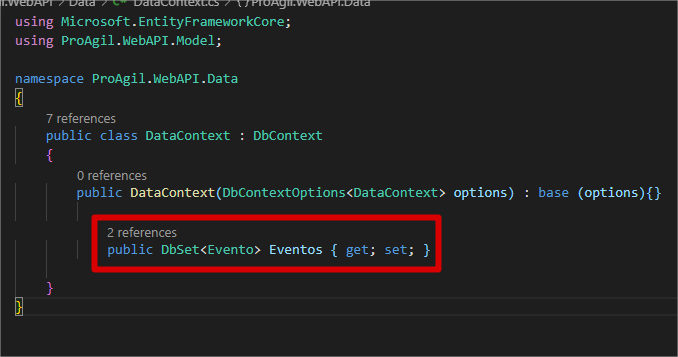
\*\***Criando Projeto(Back-End)**

* Baixar Programas:
  + Visual Studio Code
  + Dotnet Core 2.2
  + Postman (programa onde podemos testar as chamadas. Nele podemos simular as requisições)
  + SqlLite

**\*\*Configurando Ambiente e Comandos do Projeto .Net (Back-End)**

* No console executar os seguintes comandos:
  + dotnet new webapi -n nomeDoprojeto (iremos criar o template webApi porque a configuração dele é mais completa em relação a estruturado projeto).
  + D
* Baixar Extensões:
  + C#, C# Extensions, Bracket Pair Colorizer, Material icon Theme, Path Intellisense, Nuget Package Manager, vscode-icons, typescript hero
  + D
* **Comandos Básicos:**
  + **dotnet** run (roda o sistema)
  + **dotnet** clear (limpa memoria)
  + **dotnet** build (bilda o sistema)
  + **dotnet** add nomeProjetoQueTeraReferencia reference nomePastaQueDaraReferencia (Adiciona referencia nas pastas. O primeiro nome da pasta é a que irá receber a referência e irá enxergar o outro projeto)
  + **dotnet** sln nomeProjetoQueTemSolucao add nomeProjeto (criará a solução do projeto no projeto que tem a solução main)
  + **dotnet** new classlib -n nomePastaInterface (Cria uma pasta de Interface)
  + **dotnet** ef migrations add nomeScript (Irá criar a pasta Migrations com seus scripts. Com isso, poderemos no controller/service fazer consultas no banco.)
* **Instalando e Configurando Entity Framework** 
  + Passos:
    - Criar pasta Data -> Criar classe DataContext  
      
    - Dentro da classe StartUp.cs deve configurar o DbContext.  
      **Obs**: Tem que instalar o Sqlite:  
      Apertar Ctrl + Shift + P 🡪 Digita: Nuget Package Manager e depois aperta “ENTER”. 🡪 Digita: Microsoft.EntityFrameworkCore.Sqlite e depois aperta “ENTER” e seleciona Microsoft.EntityFrameworkCore.Sqlite . 🡪 Instalar a versão 2.2.0.  
        
      As vezes precisa dar um clean e build ou restore para detectar a instalação do Sqlite.  
        
      
    - Ir na classe appsettings.Development.json e criar a ConnectionString.  
        
      
    - Dentro de DataContext cria a propriedade.  
      **Obs**: Essa propriedade será o nome da sua tabela após usar o **migrations**.  
        
      
    - Criar a pasta Migrations, com isso poderemos usar o banco de dados. Lembrando que após o comando, será criado no banco todas as campos das propriedade do model que foi configurado na classe DataContext. E o nome da tabela sera o nome da propriedade criada na DataContex. Comando para criar o Migrations:   
      **dotnet ef migrations add nomeScript . Depois: dotnet ef database update (irár criar o arquivo .db)  
      Obs:** Se no model você criar uma propriedade com Id no final, o Entity irá detectar automaticamente que essa propriedade será a **PK.** Ex: EventoId.  
        
      Obs2: Para criar a Migrations com referência a algum projeto deve estar na pasta do projeto que quer, no caso aqui é o repositor e digitar: dotnet ef --startup-project ..\NomeProjetoStartUp\ migrations add nomeMigration
    - Para criar uma chave estrangeira ( FK ) de colocar o nome da entidade +Id no final. Ex: EventoId, LoteId. Dessa forma, o Entity reconhece que é uma chave estrangeira.

**\*\*Configurando Ambiente e Comandos do Angular (Front-End)**

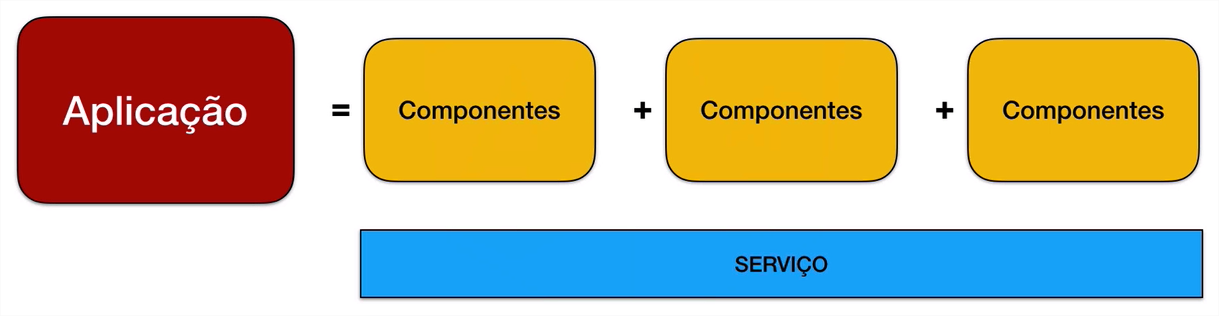
* No console executar os seguintes comandos para instalar:
  + npm install -g @angular/cli (instala o Angular Cli)
  + ng new nomeProjeto (Cria um novo projeto. Aparecerá se vai querer criar as rotas e você escolhe “sim” caso queira já criar automaticamente)
  + ng serve (Roda o sistema)
  + npm install --save bootstrap @fortawesome/fontawesome-free
  + npm install ngx-bootstrap --save (Para poder usar tooltipe, dropdown, modal...) Lembrar de configurar no app.module.ts ( import { BsDropdownModule} from ‘ngx-bootstrap’) 🡪 Em imports colocar: BsDropdownModule.forRoot()
* Baixar Extensões:
  + Angular Files, Angular Language Service, angular2-switcher, Auto Rename Tag, Bracket Pair Colorizer, c# Extensions, Debugger for Chrome, Live Server, Material Icon Theme, npm, Nuget Package Manager, Path Intellisense, Prettier – Code formatter, TSLint , vscode-icons, typescript hero, TypeScript Extension Pack, TypeScript Importer

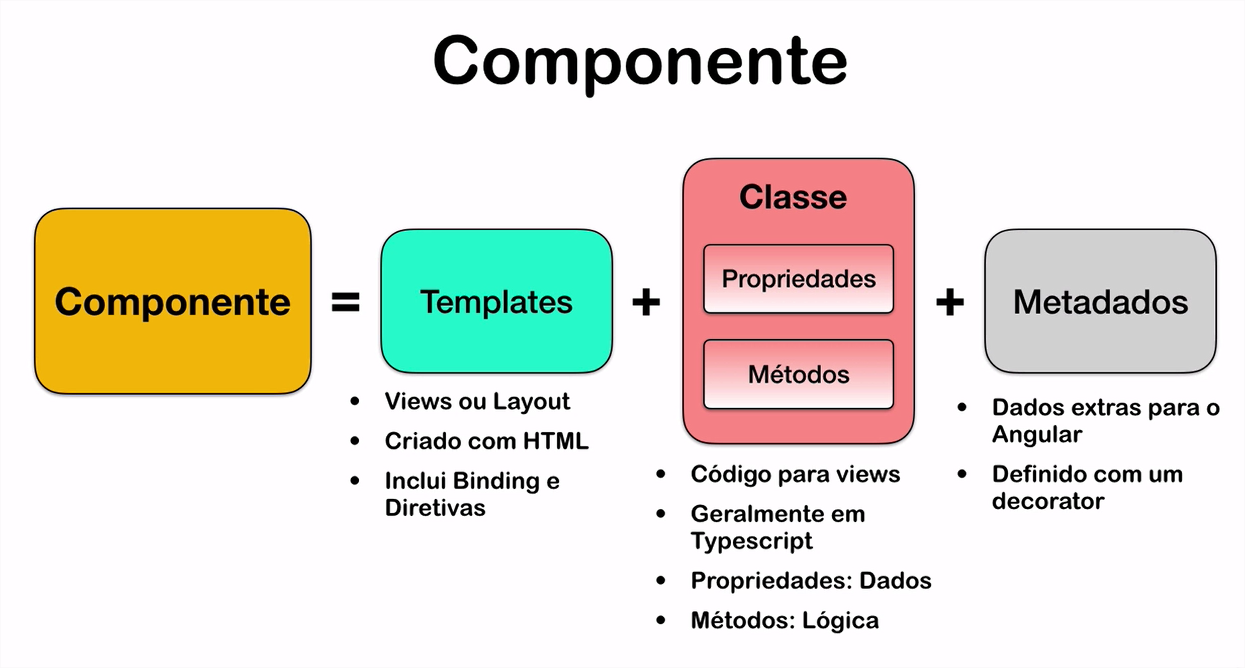
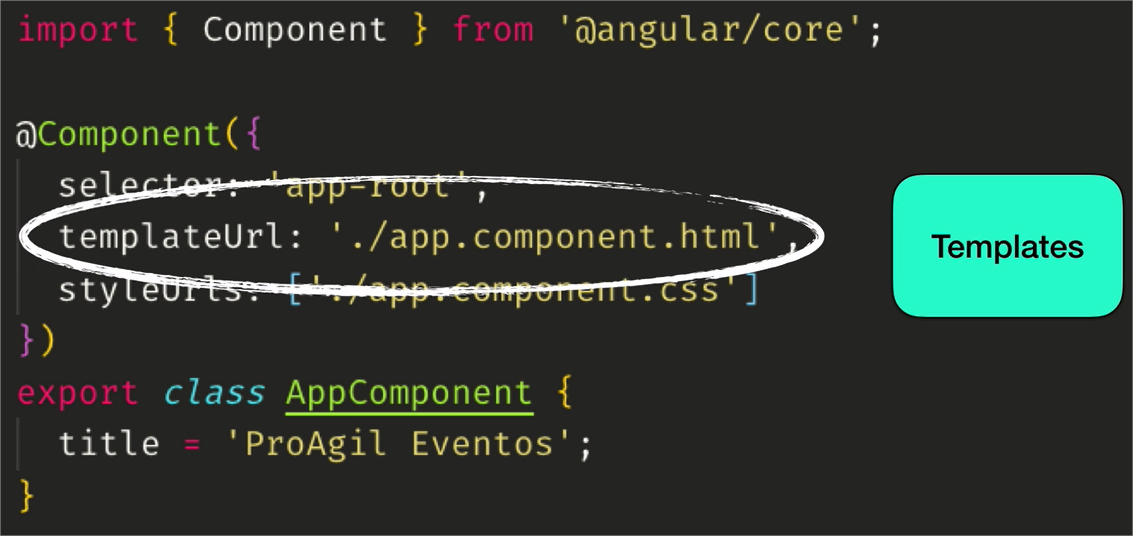
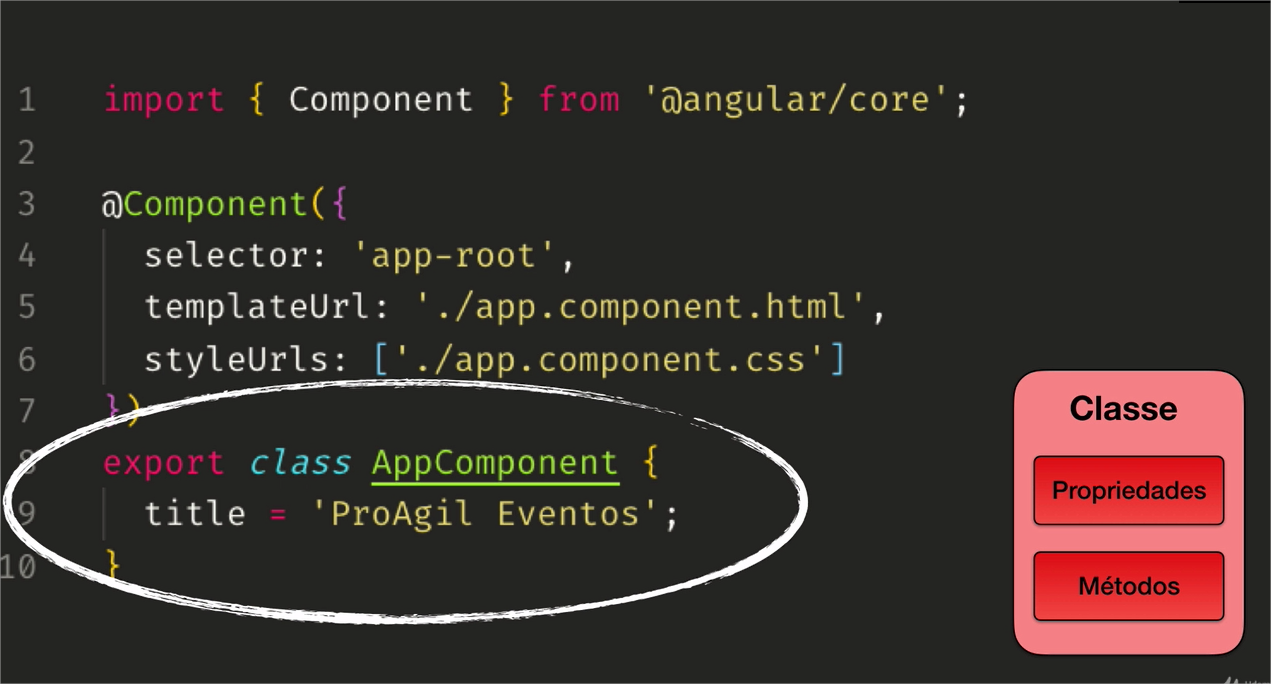
**\*\*Informações do Ambiente Angular**

* **Pastas e Arquivos:**
  + A pasta e2e é votada mais para teste unitario.
  + .editorConfig 🡪 É um arquivo de configuração. Pode ser usada com uma extensão(Prettier) para formatação do editor da extensão, como tamanho da letra, estaçamento...
  + .angular.json 🡪 Faz a configuração e o relacionamento de arquivos. Uns grandes exemplos são a configurações do script e o styles.
  + Package.json 🡪 Possue todas as dependencias do projeto angular. Com esse arquivo, podemos excluir a pasta node\_modules. Pois a pasta modules possue as dependências tambem, porem com o npm install no Package.json já baixa todas as dependências necessárias.
  + Tsconfig.json 🡪 Detalhes da configuração do angular. (Ex: “baseUrl”: “./” é onde configura a raiz do prjeto.)
  + Tslint.json 🡪 Configuração dos espaçamentos, questões dos padrões do JS.
  + Pasta SRC possue a raiz do projeto.
    - Dentro do SRC existe uma pasta APP, onde dentro dela existem 6 arquivos.
      * App.component.ts 🡪 é um arquivo onde criamos componentes que basicamente são classes em typescript. Esses componentes renderizam HTML.
      * App.component.html 🡪 Html..
      * App.module.ts 🡪 Todos componentes criado no angular tem de ser referenciado dentro dele. incialmente possui o primeiro modulo do angular, mas também serve para carregar todos os módulos do angular e seus componentes.
      * App-routing.module.ts 🡪
      * App.component.css 🡪 CSS...
      * App.component.spec.ts 🡪 Arquivo de teste unitário.

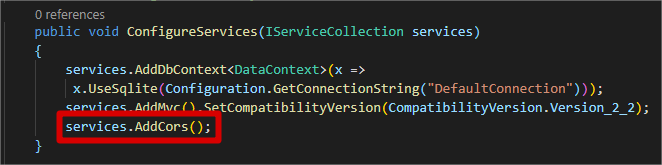
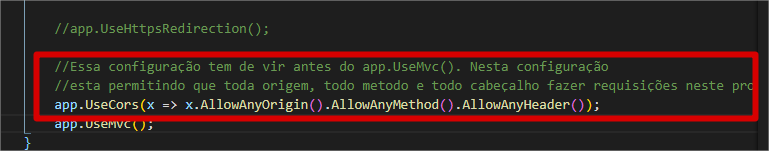
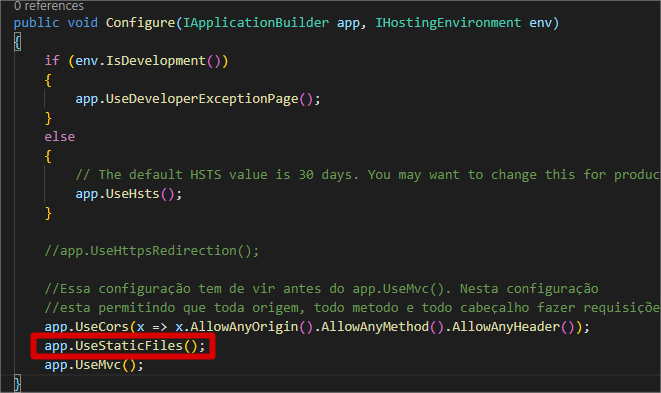
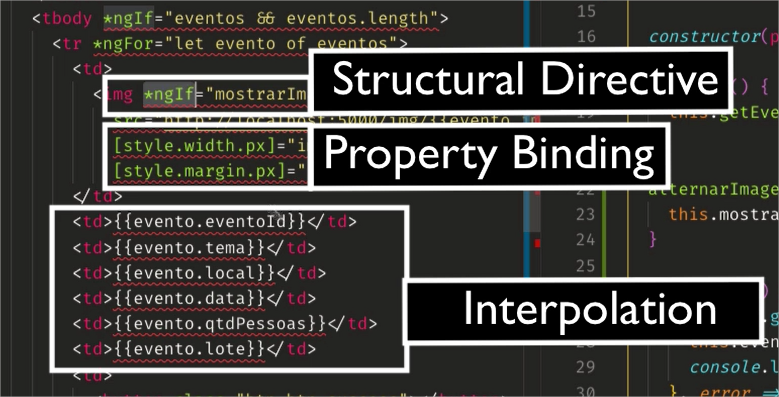
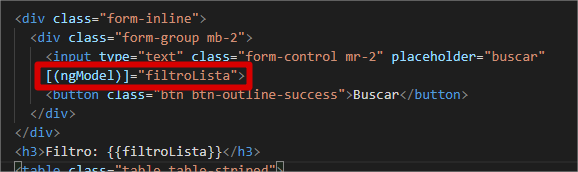
Obs. 1: Todo componente angular vai importar o componente principal que é o Component .

* **Aplicação** possui vário componentes.



* **Componente** basicamente possui **Templates** (views ou Layout que são criados pelo HTML. Que inclui Binding e Diretivas) + **Classe** (Código para as views que são geralmente em TypeScript + propriedades e métodos) + Metadados (Dados extras de uma classe para o angular. E é definido como um decorator)  
    
    
     
    
     
  O metadados é o @Component. Não é possível ter um componente sem decorar sua classe com esse metadado.

\*\* **Observações Importantes**:

* Para que no angular possa fazer uma requisição no projeto WebApi, tem que configurar no arquivo Startup.cs.   
    
    
  
* Para a API disponibilizar imagens dentro do servidor, tem que configurar no arquivo Startup.cs. As imagens ficam dentro da pasta wwwroot e dentro desta pasta criamos outra pasta chamada de img. Essas pastas devem ser criadas no projeto WebApi e não no projeto do angular.  
  
*   
    
  Obs. 1: O **two-way** só funciona se configurar no app.module.ts. Tem de importar o FormsModule. (Import { FormsModule } from ‘@angular/forms’. E também não funciona em um form simples. Qualquer coisa, coloca em uma div.  
    
  Obs. 2: O interpolation pode usar filtro por meio do pipes com o símbolo “ | “.   
  Ex: <td>{{ evento.tema | lowercase }}</td> deixará tudo em letra minúscula.
* **TWO-WAY Data Binding’**
* Observable: