**Curso da Glaucia Lemos:**  
Asp.Net Core com Entity Framework e Angular JS. Note que eu estou Adaptando esse Curso para o meu Software SUPER SUPORTE. Ao invés de fazer Filmes (como é o exemplo dela) irei já fazer um sistema prático - SuperSuporte, na pasta c:\PdvDev\SuperSuporte.

**Notas sobre o Visual Studio Code:**  
Instalei o Plugin C# (buscar addons, é da Microsoft).  
Instalei o Plugin Entity Framework.  
Instalei o Plugin GitHub Pull Requests (da GitHub, permite fazer checkin)  
Além disso instalar os Seguintes Plugins: C# (ms-vscode), .run Code Runner (formulahendry), C# FixFormat (Leopotam), C# Extensions (jchannon).  
Entre no diretório do projeto, digite no prompt code . e ele abrirá já no diretório (workspace).  
O Visual Studio Code tem um Terminal Embutido (View>Terminal) (Windows Powershell) onde pode rodar comandos.  
Você também pode entrar no View > Output (pra ver as saídas) e habilitar na mesma tela o Terminal.  
CTRL+P você pode pesquisar um arquivo, dar ENTER e abrir ele direto.

**Nota** se eu usar o Visual Studio 2017 Community para criar um projeto do tipo Asp Core: Já dá pra botar Autentication e funciona show demais!

**Notas sobre Prompt de Comando:**  
Usei o Prompt de Comando "cmDer" (está na pasta Dev\Vs2017).  
dotnet --version (mostra versão)  
dotnet new razor -o NomeProjeto (cria um projeto inteiro em Asp.Net Core)  
dotnet run (na raíz) roda o projeto  
dotnet build (na raíz) somente compila pra ver se está tudo certo  
dotnet restore (dá um rebuild na aplicação e fala se está tudo OK)  
dotnet sln add SuperSuporteTest\SuperSuporteTest.csproj (comando pra adicionar um PROJECT a uma SOLUTION).  
  
**Entity Framework Core:**  
add package (adiciona pacote)  
dotnet ef migrations add BaseInicial (gera o código para criar o banco de dados inicial no context – dar o nome que quiser)  
dotnet ef database update (roda o EF e cria o banco de dados – roda o migration)  
Dica Fernando: Apague todos os arquivos principalmente o “SuperSuporteContextModelSnapshot” que ele cria dentro da pasta Migrations, senão ele não cria as tabelas novas, fica bugadão. Fica recriando tabelas antigas, não dá pra entender. Ao rodar o Update, apague todos arquivos velhos. Também cuidado com o uso do banco, se estiver aberto no Management (amarrado) ele não consegue atualizar e dá uns erros loucos.  
  
**Scaffolding (criação de Razors/Views automáticas):**  
dotnet tool install --global dotnet-aspnet-codegenerator --version 2.1.1 (para saber a versão certa, vá em VsCode>launch.json e veja netcoreapp)  
dotnet aspnet-codegenerator razorpage -m Filme -dc FilmeContext -udl –outDir Pages\Filmes -referenceScriptLibraries   
(Explicação: código para gerar automaticamente as páginas em Razor - scaffolding) (Filme é o Model, FilmeContext é o arquivo de Contexto, e nesse caso, ele irá criar as páginas dentro de Pages\Filmes, poderia ser o Model “Cadastro” e a pasta “Pages\Cadastros por exemplo).  
  
Nota: Após rodar esse comando, fechar o cmder e abrir de novo.  
Nota 2 Linux: Pra Linux a ‘barra’ seria ao contrário Pages/Filmes – só isso.  
Nota 3: Crie a pasta em Pages\Empresas no plural, nunca com o nome do model, senão vai conflitar.  
Exemplo Uso: **dotnet aspnet-codegenerator razorpage -m Empresa -dc SuperSuporteContext -udl -outDir Pages\Empresas –referenceScriptLibraries  
  
Dicas sobre o Identity (Autenticação Automática) do Asp.Net Core:**Para customizar as Razors de Login, você deve clicar no Projeto com o Direito > ADD > New Scaffolding Item. Ai selecione “Identity” e o Visual Studio irá entender que você quer customizar as Views de Login. **Nota sobre as validações automáticas:** Aquelas mensagens automáticas que aparecem (This Field is required) na View que vem lá do Required feito nos Models (data annotation), irão aparecer em inglês. Para traduzir procure o jquery.validate.js (só mudar a tradução nesse arquivo). Procure “Field is required” dentro desse arquivo e vá traduzindo para português.  
**Ordem dos comandos usando Nuget para instalar Addins na Aplicação (rodar no Cmder na raíz do projeto):**dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design (necessário para o design do EF core)  
dotnet add package Microsoft.EntityFrameworkCore.Sqlite (instala o SQL Lite, esse não rodei, estou usando SQL Server no projeto).

**Notas sobre Desenvolvimento:**  
.gitignore (o arquivo na raiz do projeto diz quais arquivos serão ignorados no commit)  
Instalei o .Net Core e o .Net SDK que estão atualizados na pasta Dev\Vs2017.  
Nuget está para o .Net assim como o NPM está para Node / Angular.  
Code Generator é uma ferramenta para criar as RAZORS (Interfaces) de acordo com os Models (parecido com Scaffolding).  
Nota: É preciso adicionar a seguinte linha no Startup.cs (ConfigureServices) para que o programa ache o Context (string no appsettings.json):  
services.AddDbContext<SuperSuporteContext>(options => options.UseSqlServer(Configuration.GetConnectionString("SuperSuporteConnection")));  
  
**MSTest (Testes Unitários):**dotnet new mstest --force (na raiz do projeto, o “—force” (dois traços) força a atualização do csproj, arquivo do projeto, ele será recriado adicionando as bibliotecas necessárias, e criando um “UniTest1.cs” na raiz do projeto onde estarão os testes. Nota: Eu precisei criar o projeto fora da pasta, e depois integrar as coisas e copiar o .cs (do test) para o projeto. Nota2: Precisei adicionar linhas no .csproj, tanto no PropertyGroup como no ItemGroup, o arquivo inteiro ficou assim:  
  
<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>

<TargetFramework>netcoreapp2.1</TargetFramework>

<GenerateProgramFile>false</GenerateProgramFile>

<UserSecretsId>aspnet-SuperSuporte-0F57B735-CA57-4D9B-B46C-C7F121025A5C</UserSecretsId>

</PropertyGroup>

<ItemGroup>

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

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

<PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="2.1.5" PrivateAssets="All" />

<PackageReference Include="Microsoft.NET.Test.Sdk" Version="15.9.0" />

<PackageReference Include="MSTest.TestAdapter" Version="1.3.2" />

<PackageReference Include="MSTest.TestFramework" Version="1.3.2" />

</ItemGroup>

<ItemGroup>

<Folder Include="Areas\Identity\Services\" />

<Folder Include="Migrations\" />

<Folder Include="Models\Cadastros\" />

<Folder Include="Models\Chamados\" />

<Folder Include="Models\Usuarios\" />

<Folder Include="Pages\Chamados\" />

</ItemGroup>

</Project>

**Sobre a Estrutura do Projeto:**wwwroot: é onde ficam todos as extensões do projeto - Bootstrap, jquery, angular, node, tudo deve ficar nessa pasta.  
pages: é onde fica toda a estrutura de páginas do lado cliente (ele gera um padrão, tipo um sccafold) (razors)  
Startup.cs: é um "orquestrador" do aplicativo. Configurações, autenticação e tudo mais ficam nele.   
Pasta .vsCode: O arquivo tasks.json é onde a aplicação gera todos caminhos, e o launch é o que vai carregar.  
appsettings.json: A Connection String do Banco fica nesse arquivo.  
  
Anotação para Curso de Azure:  
\* Doc Oficial Microsoft pra publicar no Azure Direto do Git: https://docs.microsoft.com/pt-br/azure/app-service/app-service-deploy-local-git  
\* VSCode - Baixar a Extensão Azure App Service para integrar.  
\* Serviços de Aplicativos (onde você armazena seu site)  
\* PowerShell – habilitar deploy (irá criar no serviços de aplicativos um nome de implementação) - comando (executar no powershell dentro do site): az webapp deployment user set --username fernandopassaia --password @1234fd@  
\* Kudu: HTTPS://<seusite>.scm.azurewebsites.net (mostra informações importantes sobre seu site)