MODULO 2 – ASP .NET CORE

PROJETO – DEVFREELA
- Plataforma para cadastro e contratação de serviços de Freelancer

FUNCIONALIDADES IMPLEMENTADAS:
- Cadastro, Atualização, Cancelamento, Consulta de Serviços;
- Cadastro de Usuarios, Perfis Freelancer e Clientes;
- Adicionar comentários para um serviço Freelancer;
- Definir, Iniciar e Finalizar o Projeto;

1 - QUAL O PROBLEMA
- Foco sempre atacar o problema, primeiro passo é entender o domínio que ataca o problema e o resolve;
PROBLEMA:
- Alta demanda de novos projetos, necessidade de delegar módulos e projetos inteiros;
- A equipe da empresa esta focada em projetos internos;
- Os prazos potencializam a necessidade;

2 - MODELANDO O DOMINIO
- Definir a solução mais fiel possível do ponto de vista do usuário. Para entender o que ele quer de verdade;
- Analisar os Substantivos envolvidos, pois possível se tornarão Entidades e os Verbos são possíveis candidatos;

3 - NARRATIVA DA API A SER CRIADA
- DevFreela oferece uma plataforma unificada, para o cliente contratar os melhores Freelancers para seus projetos de maneira confiável;
- Os clientes definem as características dos projetos que publicam. Os freelancers se comunicam com eles a respeito do projeto e suas habilidades;
- Clientes iniciam o projeto com o freelancer que tenha um perfil adequado ao projeto, interagindo(Mensagem) com ele ate concluir o projeto; Iniciar, Concluir, Interagir = Candidatos a se tornarem funcionalidades do projeto;

4 - ENTIDADES EXTRAIDAS DA NARRATIVA:
- Clientes / Freelancers;
- Projetos;
- Habilidades;
- Mensagens entre clientes e freelancers;
AÇÕES EXTRAIDAS
- Definir, Iniciar, Terminar o projeto; CRUD do projeto
- Cliente e Freelancer se comunicarem;
- Definição das habilidades de um freelancer;

PROTOCOLO – HTTP
- Base para comunicação de dados na internet; Quando desenvolvemos uma API permitimos o acesso aos recursos através do HTPP;

CABEÇALHOS REQUIÇÕES HTTP
Os cabeçalhos de requições HTTP, tanto de envio quanto de respostas, permitem o envio de informações extras, os cabeçalhos mais comuns são:
- Content-Length – Tamanho da requisição;
- Conten-Type – Tipo da requisição que espera receber;
- Cache-Control – A estratégia de cache, que se deve ou não salvar;

STATUS HTTP – São códigos de respostas de conclusão de uma requisição. Os mais comuns são:
- 200 – OK;
- 201 – Created: Requisição de sucesso e resultou na criação de um recurso. EX: Cadastro um usuário retorna 201;
- 202 – Acepted: Requisição de sucesso mas aguarda um processamento. EX: Uma fila de mensagens, processo de pagamento pedido foi confirmado;
- 204 – No Content – Requisição de sucesso mas não tem resposta necessária. EX: Atualização de objetos e remoções de objetos;

- 400 – Bad Request: Algum dado que você enviou não esta correto. EX: Não pode passar de 1000 caracteres;
- 401 – Unauthorized: A requisição não foi aprovada do ponto de vista do usuário. EX: Não esta autorizada com usuário e senha;
- 404 – Not Found: O recurso da requisição não foi encontrado. EX: Consulta de um trabalho passando um ID mas não existe;
- 500 – Internal Server Error: Erro aleatório não tratado em nosso sistema;

METODOS E VERBOS – HTTP
Representam ações para um dado recurso(ação), mais comuns são:
- GET: Uma consulta ou ação que não altera o estado do sistema, geralmente retorna 200; Apenas suma consulta não altera o estado do sistema;
- POST: Criação de recurso, retornando 201 ou 400; cadastro novo usuario, novo projeto;
- PUT: Atualização de recurso, retorna 204, 400, 404;
- DELETE: Remoção de recurso, retorna 204, 404; Cancelar um projeto;

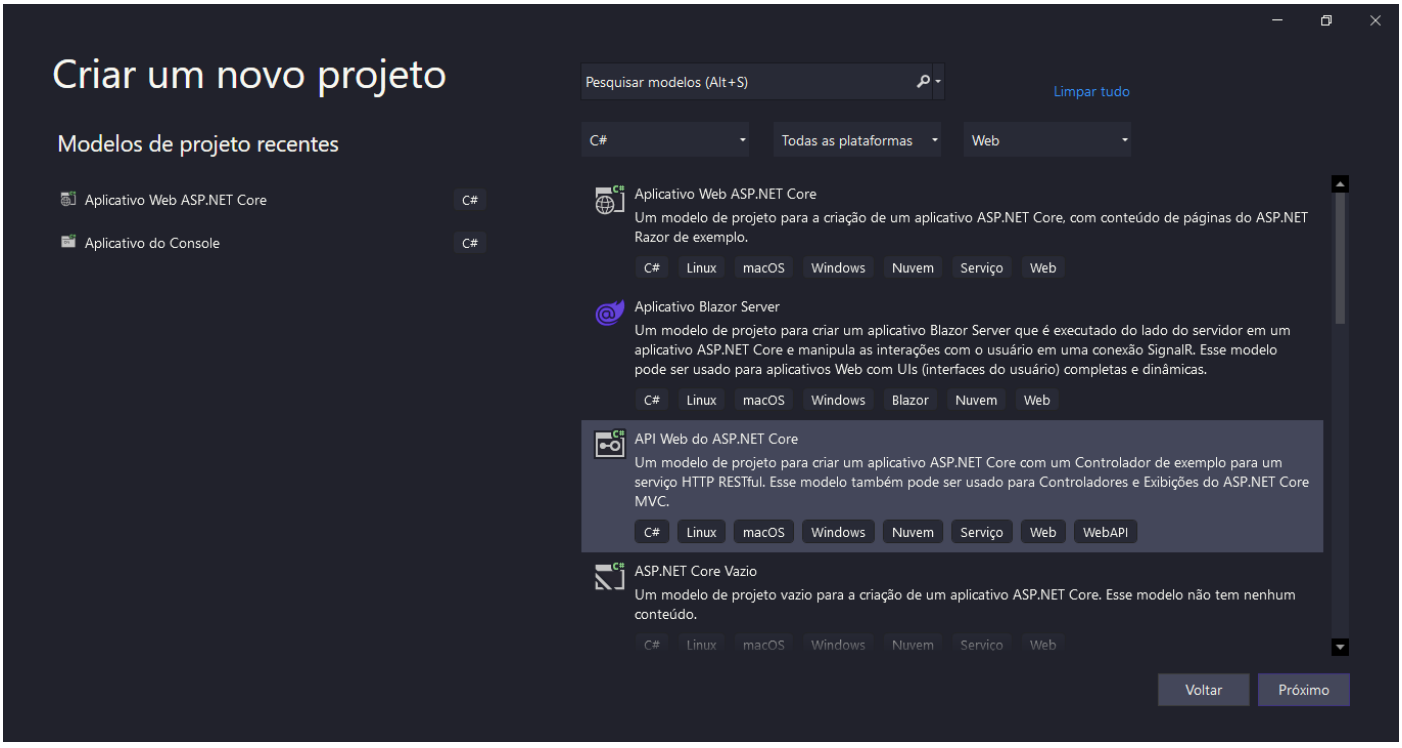
API REST
- REST(Represent State Transfer): É um padrão de comunicação entre sistemas, usado geralmente junto ao HTTP; Baseado em recursos(Entidades da API), que são representados em um caminho de acesso URL; Tem uma serie de modelos padrões;

EXEMPLOS DE API REST
URLs seguindo o padrão REST para acesos de recurso:
- api/users/1: GET, PUT, DELETE – Podemos consultar, atualizar ou deletar o usuário;
- api/projects: GET, POST – Refere-se a um conjunto de usuários, projetos. Ou usar Post para cadastrar
- api/skils: GET, POST
- api/users/1/projects: GET, POST – Podemos utilizar mais de um recurso na Url, acesso recursos e projetos do usuário;

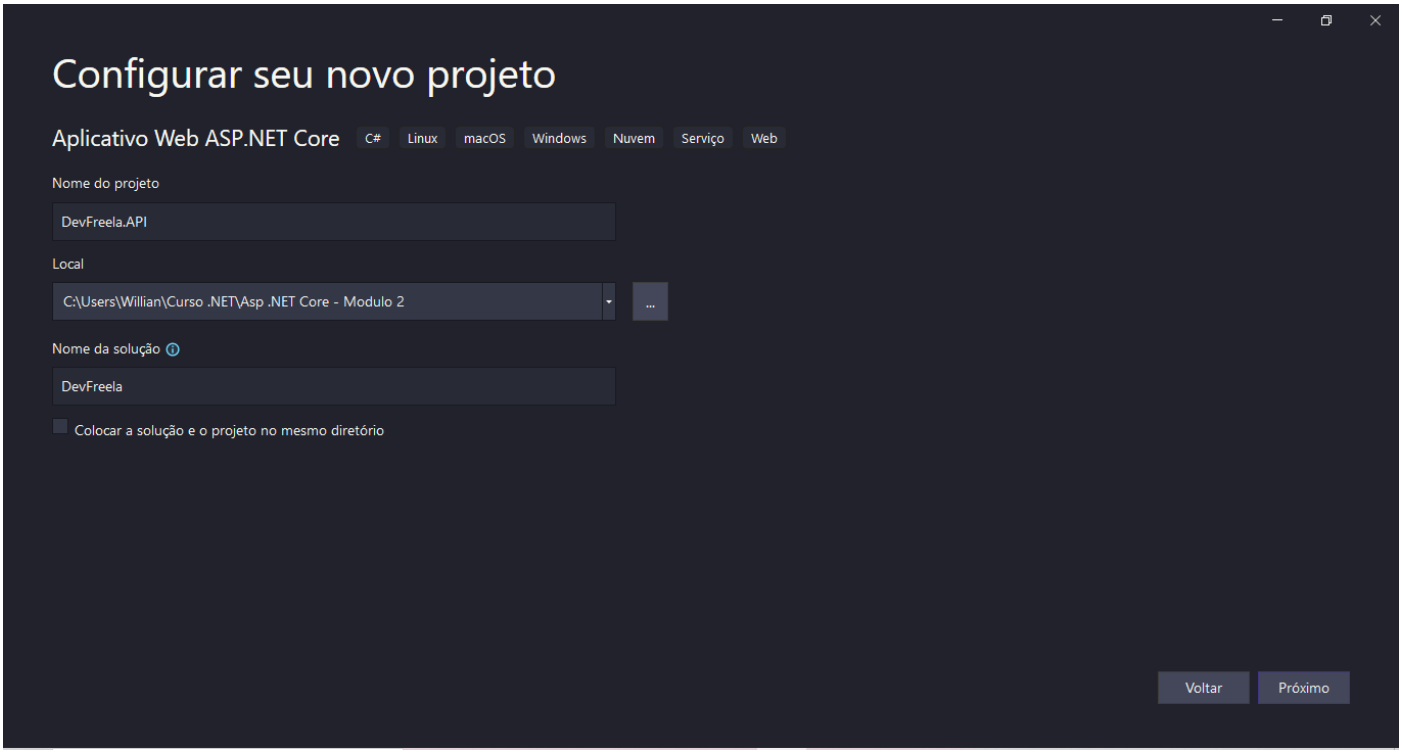
ASP.NET CORE

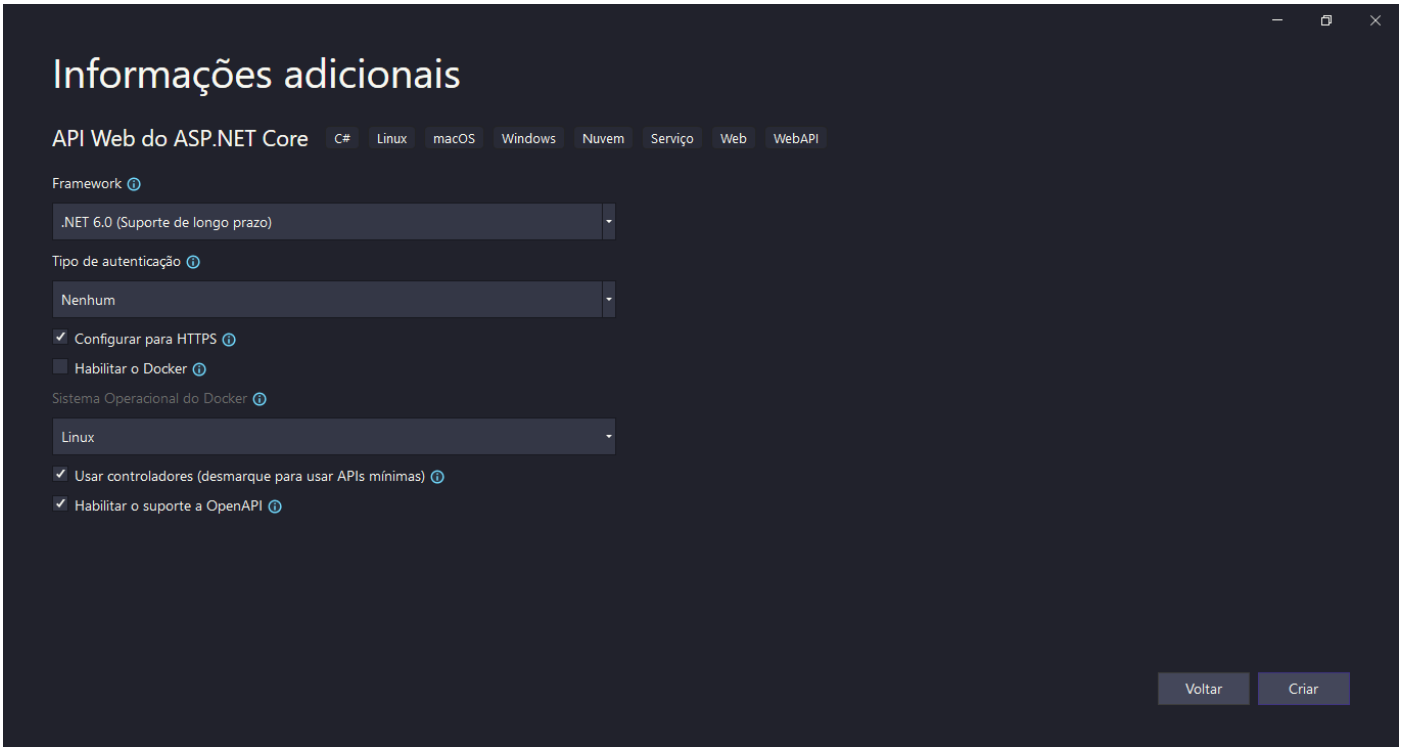
Framework open-source, recebe melhorias da comunidade; Multiplataforma, leve e focado em alta performance; Principal opção para desenvolvimento .NET;
Contém um sistema próprio de configuração baseado em ambientes, facilitando o uso em nuvem; Possui injeção de dependência nativo, sem usar bibliotecas como o Ninject; Melhor escolha para projetos .NET;

CRIANDO PROJETO ASP .NET CORE

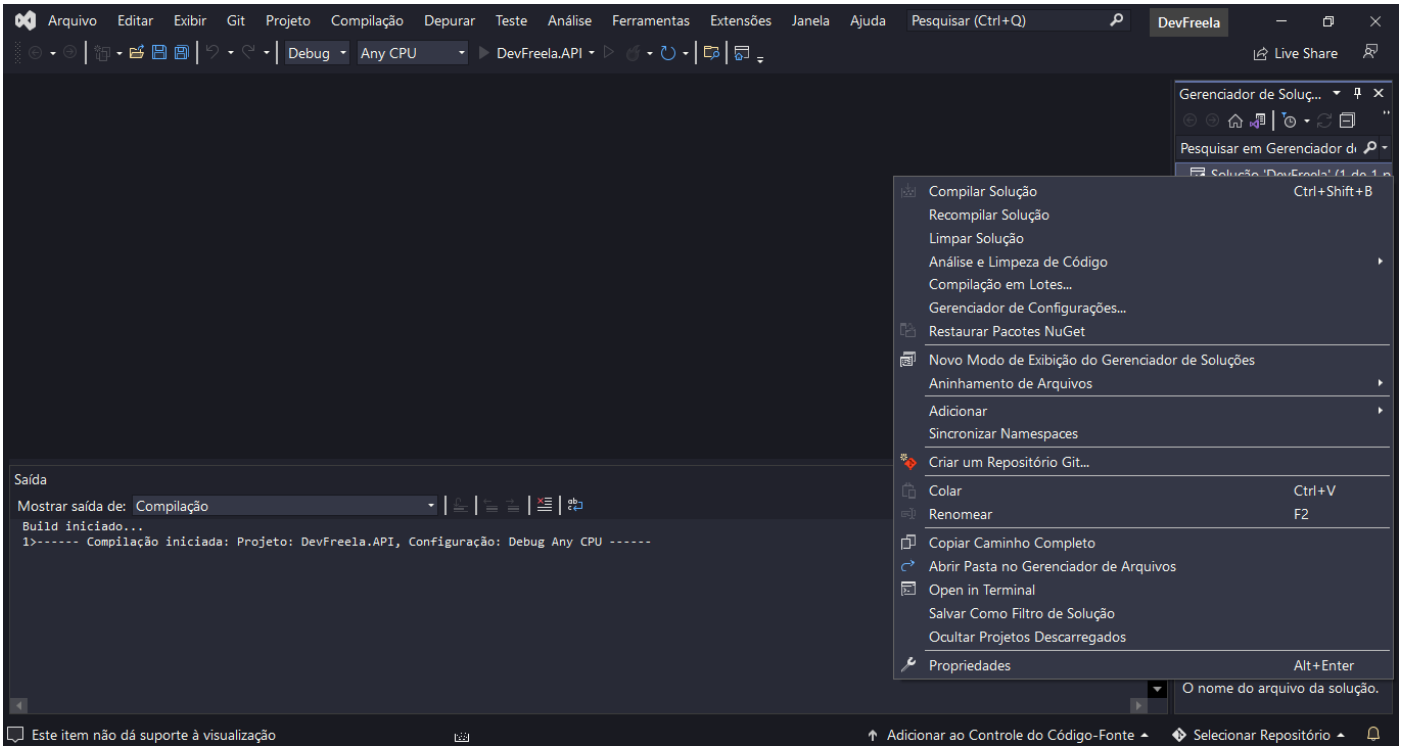


NOME PROJETO: DevFreela.API
SOLUÇÃO: DevFreela – Porque a solução é como se fosse um conteiner de projetos, como o projeto terá varias camadas vamos usando o .API para ir separando;





COMPILAR A SOLUÇÃO: Clicar com o botão direito e compilar



CRIANDO PROJETO VIA LINHA DE COMANDO

Criar a pasta do projeto:

\dotnet new -- help ou \dotnet new –help (Para ver as opções de criação de projetos)

Selecionar Prompt de Comando

```
C:\Users\Willian\Curso .NET>cd teste

C:\Users\Willian\Curso .NET\teste>dotnet new -- help
O comando 'dotnet new' criar um projeto .NET baseado em um modelo.

Os modelos comuns são:
Nome do modelo              Nome Curto    Idioma      Tags
--------------------------  ------------  ----------  -------------------
Aplicativo do Console       console       [C#],F#,VB  Common/Console
Aplicativo do Windows Forms winforms      [C#],VB     Common/WinForms
Aplicativo WPF              wpf           [C#],VB     Common/WPF
ASP.NET Core Web App        webapp,razor  [C#]        Web/MVC/Razor Pages
Biblioteca de Classes       classlib      [C#],F#,VB  Common/Library
Blazor Server App           blazorserver  [C#]        Web/Blazor

Um exemplo seria:
   dotnet new console

Exibir opções de modelos com:
   dotnet new console -h
Exibir todos os modelos instalados com:
   dotnet new --list
Exibir modelos disponíveis no NuGet.org com:
   dotnet new web --search


C:\Users\Willian\Curso .NET\teste>dotnet new --help
Usage: new [options]

Options:
  -h, --help                    Exibe a ajuda para este comando.
  -l, --list <PARTIAL_NAME>     Lista os modelos que contêm o nome do modelo especificado. Se nenhum nome for especificado, lista todos os modelos.
  -n, --name                    O nome da saída que está sendo criada. Se nenhum nome for especificado, o nome do diretório de saída será usado.
  -o, --output                  Local para colocar a saída gerada.
  -i, --install                 Instala um pacote de origem ou de modelo.
  -u, --uninstall               Instala uma origem ou pacote de modelo.
  --interactive                 Permite que o comando dotnet restore interno seja interrompido e aguarde a ação ou entrada do usuário (por exemplo, para concluir a autenticação).
  --add-source, --nuget-source  Especifica uma fonte do NuGet a ser usada durante a instalação.
  --type                        Filtra modelos com base em tipos disponíveis. Os valores predefinidos são "projeto" e "item".
  --dry-run                     Exibe um resumo do que aconteceria se a linha de comando especificada fosse executada se resultar em uma criação de modelo.
  --force                       Força o conteúdo a ser gerado mesmo que ele altere arquivos existentes.
  -lang, --language             Filtra modelos baseados em linguagem e especifica o idioma do modelo a ser criado.
  --update-check                Verifique os pacotes de modelo instalados atualmente para atualização.
  --update-apply                Verifique os pacotes de modelo instalados atualmente para atualização e instale as atualizações.
  --search <PARTIAL_NAME>       Pesquisa os modelos em NuGet.org.
  --author <AUTHOR>             Filtra os modelos com base no autor do modelo. Aplicável somente com --busca ou --lista | -l opção.
  --package <PACKAGE>           Filtra os modelos baseados na ID do pacote NuGet. aplica-se a --pesquisa.
  --columns <COLUMNS_LIST>      Lista de colunas separadas por vírgulas a serem exibidas na saída --lista e --pesquisa.
                                As colunas com suporte são: idioma, marcas, autor, tipo.
  --columns-all                 Exibir todas as colunas na saída de --lista e --pesquisa.
  --tag <TAG>                   Filtra os modelos com base na marca. Aplica-se a --pesquisa e --lista.
  --no-update-check             Desativa a verificação de atualizações de pacote de modelo ao instanciar um modelo.
```

\dotnet new --list (Mostra vários tipos de projetos, templates)

```
Selecionar Prompt de Comando
C:\Users\Willian\Curso .NET\teste>dotnet new --list
Esses modelos correspondem à sua entrada:

Nome do modelo                                 Nome Curto           Idioma      Tags
---------------------------------------------  -------------------  ----------  -------------------------
Aplicativo do Console                          console              [C#],F#,VB  Common/Console
Aplicativo do Windows Forms                    winforms             [C#],VB     Common/WinForms
Aplicativo WPF                                 wpf                  [C#],VB     Common/WPF
Arquivo de Configuração da Web                 webconfig                        Config
arquivo de dotnet gitignore                    gitignore                        Config
Arquivo de manifesto da ferramenta local Dotnet tool-manifest                   Config
Arquivo de Solução                             sln                              Solution
Arquivo EditorConfig                           editorconfig                     Config
arquivo global.json                            globaljson                       Config
ASP.NET Core Empty                             web                  [C#],F#     Web/Empty
ASP.NET Core gRPC Service                      grpc                 [C#]        Web/gRPC
ASP.NET Core Web API                           webapi               [C#],F#     Web/WebAPI
ASP.NET Core Web App                           webapp,razor         [C#]        Web/MVC/Razor Pages
ASP.NET Core Web App (Model-View-Controller)   mvc                  [C#],F#     Web/MVC
ASP.NET Core with Angular                      angular              [C#]        Web/MVC/SPA
ASP.NET Core with React.js                     react                [C#]        Web/MVC/SPA
Biblioteca de Classes                          classlib             [C#],F#,VB  Common/Library
Biblioteca de Classes do Windows Forms         winformslib          [C#],VB     Common/WinForms
Biblioteca de Classes WPF                      wpflib               [C#],VB     Common/WPF
Biblioteca de Controles de Usuário do WPF      wpfusercontrollib    [C#],VB     Common/WPF
Biblioteca de Controles do Windows Forms       winformscontrollib   [C#],VB     Common/WinForms
Biblioteca de Controles Personalizados do WPF  wpfcustomcontrollib  [C#],VB     Common/WPF
Blazor Server App                              blazorserver         [C#]        Web/Blazor
Blazor WebAssembly App                         blazorwasm           [C#]        Web/Blazor/WebAssembly/PWA
Configuração do NuGet                          nugetconfig                      Config
MSTest Test Project                            mstest               [C#],F#,VB  Test/MSTest
MVC ViewImports                                viewimports          [C#]        Web/ASP.NET
MVC ViewStart                                  viewstart            [C#]        Web/ASP.NET
NUnit 3 Test Item                              nunit-test           [C#],F#,VB  Test/NUnit
NUnit 3 Test Project                           nunit                [C#],F#,VB  Test/NUnit
Protocol Buffer File                           proto                            Web/gRPC
Razor Class Library                            razorclasslib        [C#]        Web/Razor/Library
Razor Component                                razorcomponent       [C#]        Web/ASP.NET
Razor Page                                     page                 [C#]        Web/ASP.NET
Worker Service                                 worker               [C#],F#     Common/Worker/Web
xUnit Test Project                             xunit                [C#],F#,VB  Test/xUnit

C:\Users\Willian\Curso .NET\teste>
```

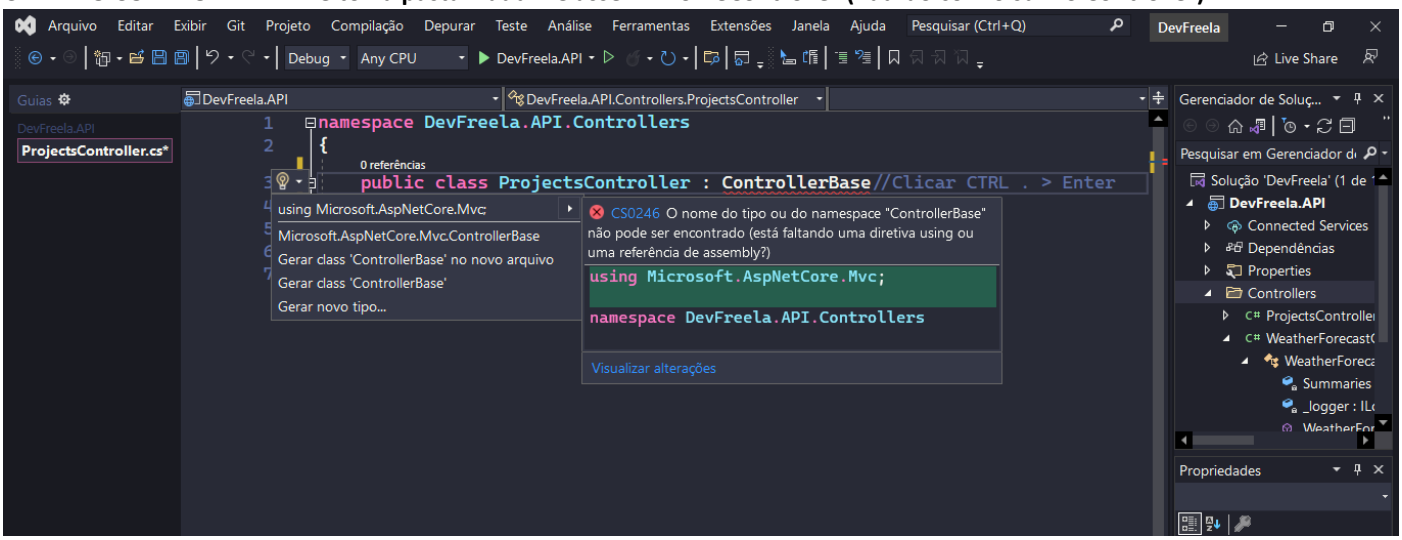
\dotnet new sln --name DevFreela (Criar primeiro a Solução a pasta sln)
\dotnet new webapi --name DevFreela.API –output DevFreela.API (Output localização onde vou por o projeto)
\dotnet sln add ./DevFreela.API/DevFreela.API.csproj (Adicionar na pasta, projeto. A referencia csproj)

CONTROLLERS e ACTIONS

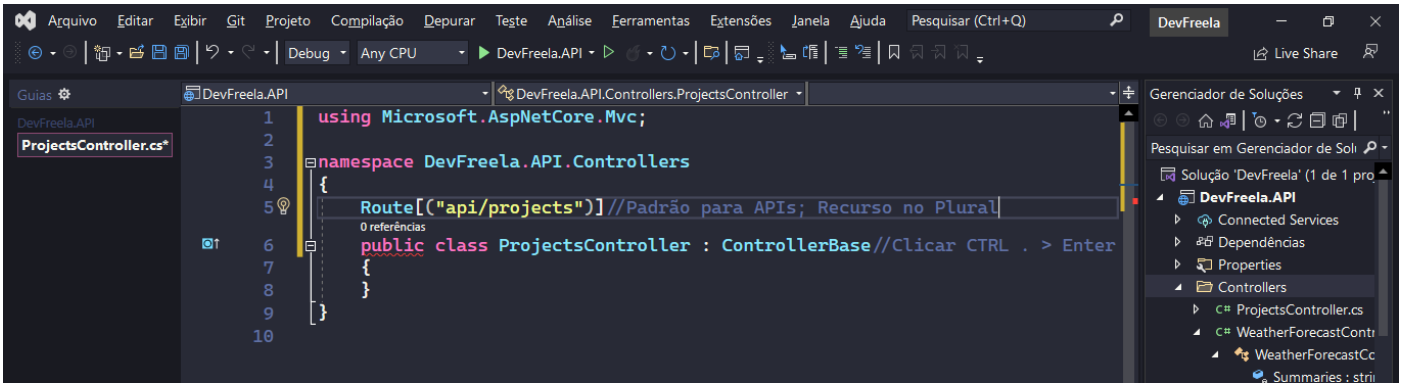
Controllers são componentes que definem e agrupam um conjunto de ações (Actions); Agrupam as ações de maneira lógica, baseado no recurso a ser tratado ou acessado; Dentro tem todas as ações;
DevFreela alguns exemplos de controllers:
- USERS: Usuarios
- PROJECTS: Projetos
- SKILS: Habilidades

Controller em uma API APS.NET CORE é uma classe que herda de ControllerBase; Uma informação importante para o Controller é a ROTA BASE para ele, já que isso influenciará na rota de todos pontos de acesso desse Controller. Ela pode ser definida pela anotação ROUTE;

CRIANDO CONTROLLER > Direito na pasta > add > Classe >> NomeController (Padrão com o sufixo Controller)



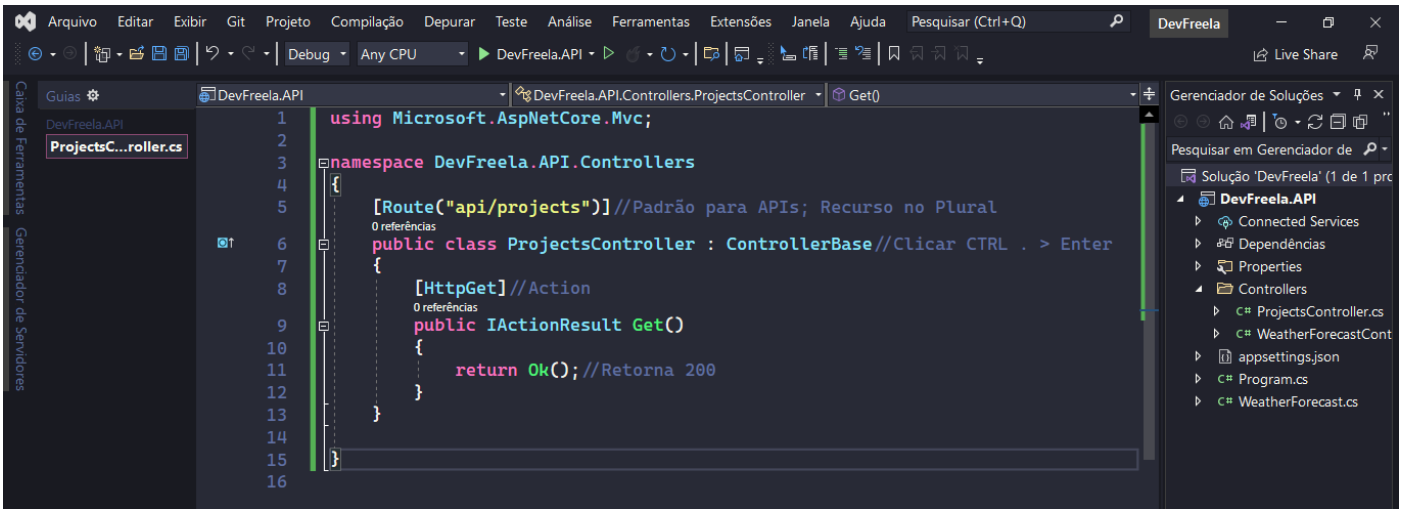
Adicionar a rota base: [Route(“api/projects”)] (Padrão para APIS o recurso Projects no plural)



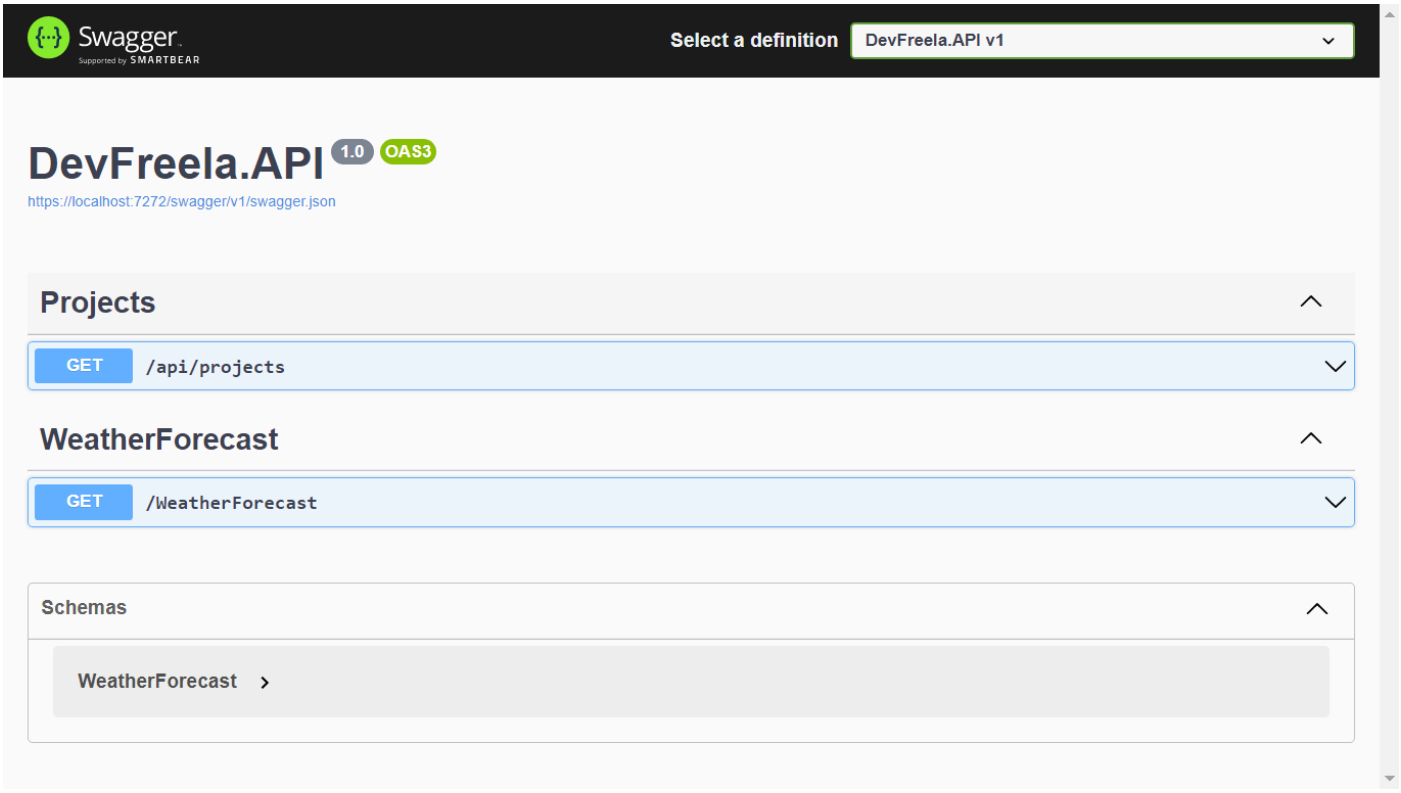
ACTIONS

São métodos contidos em uma Classe Controller e representam Pontos de Acesso de uma APIs; Para APIs o tipo de retorno de uma Action é |ActionResult > Que é uma Interface implementada pelas respostas HTTP. EX: OK, NOTFOUND...

CRIANDO ACTION (Se remover a Herança deixa de existir)



SWAGGER
É uma ferramenta que simplifica o desenvolvimento de APIs, permitindo testar, documentar...; Consegue gerar uma interface gráfica, contendo todos pontos de acessos (EndPoints) de uma API organizados por controller, permitindo realizar requisições direto de sua interface;
A compilar Controller ele abre Swagger;



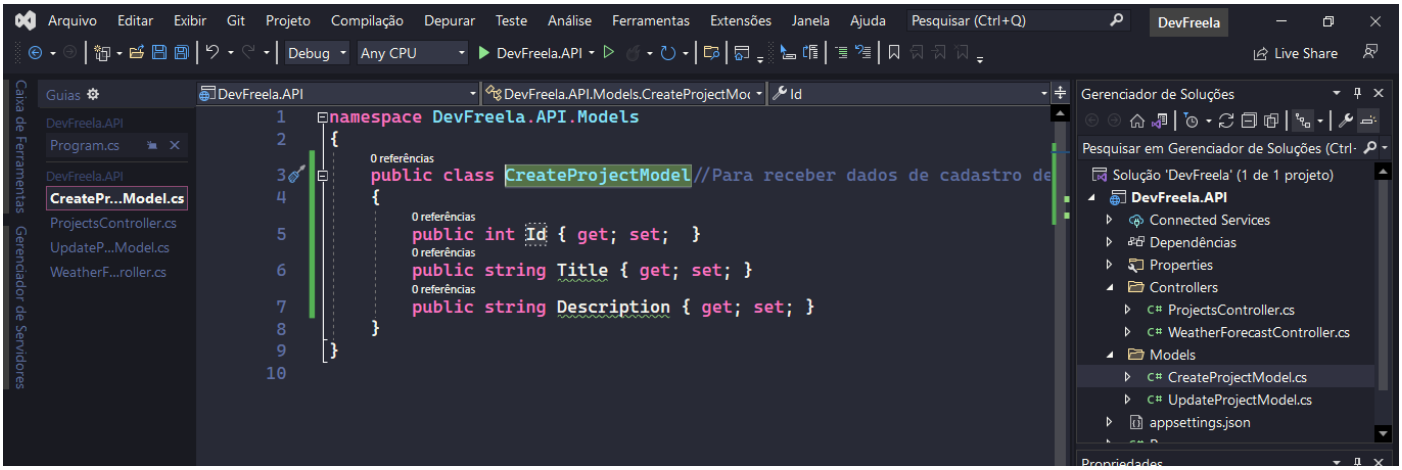
APROFUNDANDO EM ACTIONS

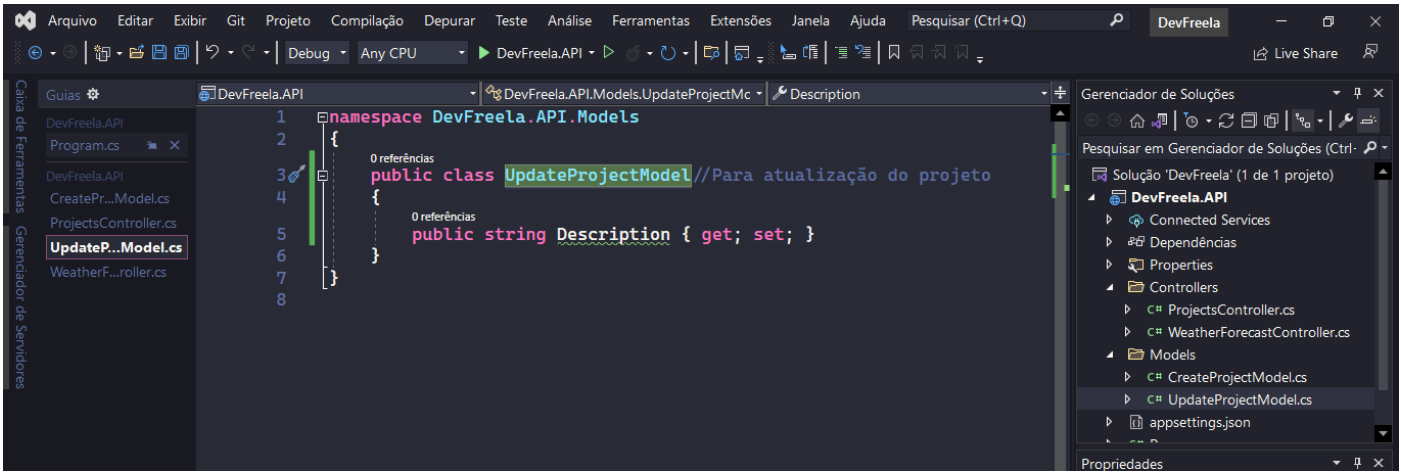
GET: Para uma requisição GET geralmente você vai receber:
- Parametros de URL: EX: api/projects/1 > O numero 1 é o parâmetro que vai ser seguido pela URL, pode ser uma consulta da APi para o projeto de identificador 1;
- Query String: Utilizada para filtros. EX: Encontrar todos projetos;

POST / PUT: Geralmente vão ser recebidos parâmetros através de:
- Parametros de URL: Realizar a atualização de um projeto usamos: api/projects/1 ou /2 > E enviamos no corpo da requisição o objeto atualizado;
- Corpo de Requisição: Cadastramos um projeto, usuário, devemos enviar pelo corpo da requisição; Para requisições POST utilizamos apenas o corpo da requisição;

DELETE: Recebemos apenas o parametro da URL EX: api/projects/1 usamos o Delete para o recusro especifico;

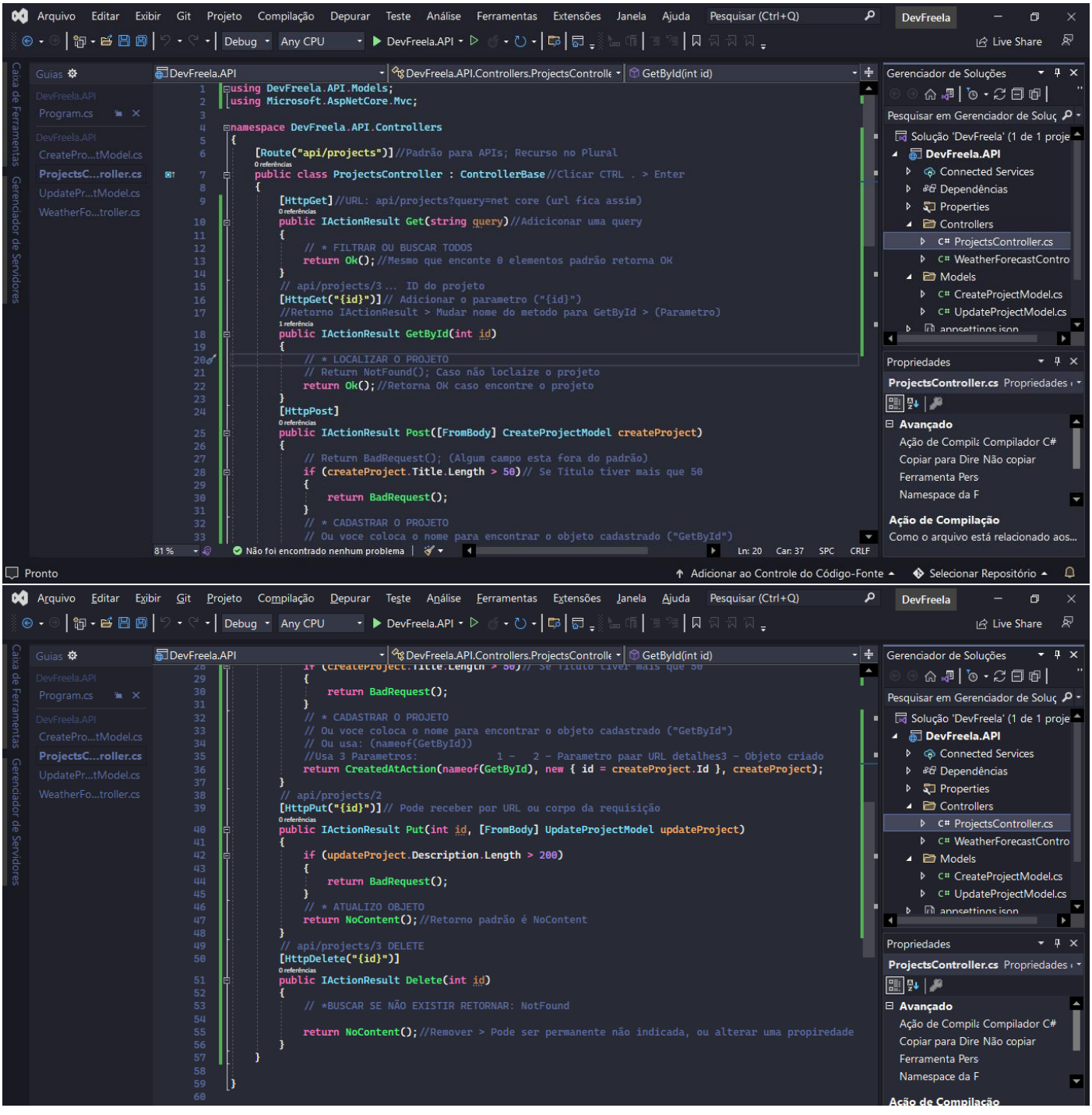
- Criar pasta MODELS > Classe CreateProjectModel (Recebe dados de ID, Titulo e Descrição) > Classe UpdateProjectModel (Para atualização do projeto)



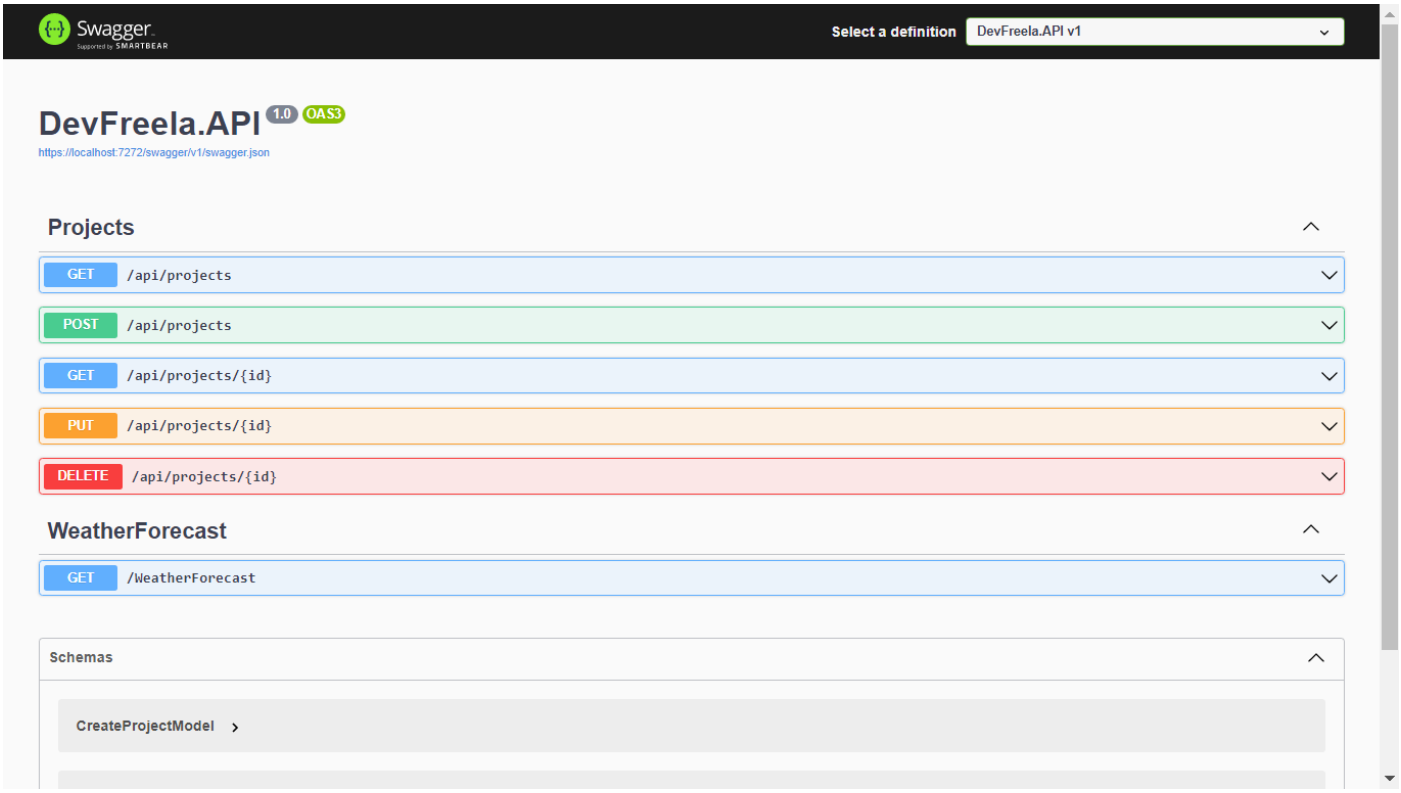


```
namespace DevFreela.API.Models
{
    0 referências
    public class UpdateProjectModel//Para atualização do projeto
    {
        0 referências
        public string Description { get; set; }
    }
}
```

- Classe ProjectsController > Adicionar uma Query (Para APIs Get podemos passar uma Query: Informações na URL)



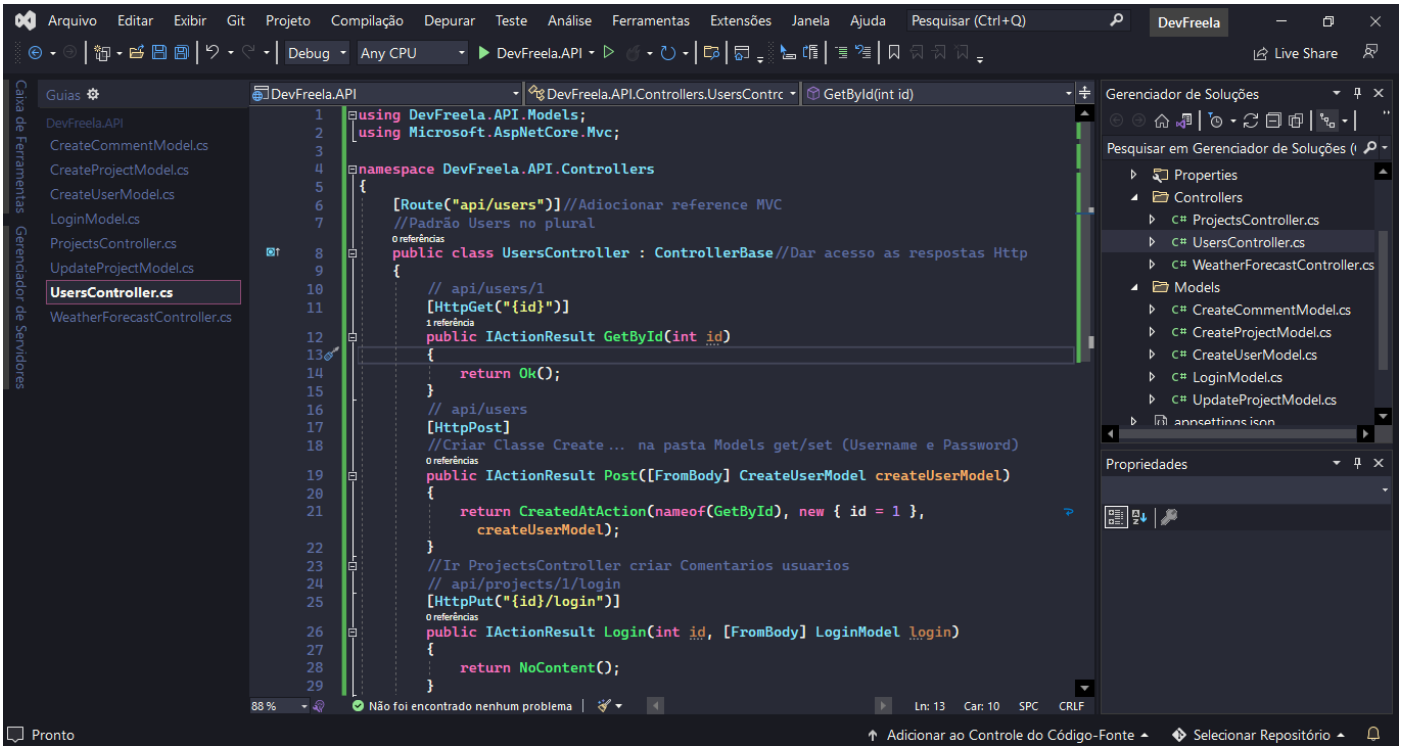
```
using DevFreela.API.Models;
using Microsoft.AspNetCore.Mvc;

namespace DevFreela.API.Controllers
{
    [Route("api/projects")]//Padrão para APIs; Recurso no Plural
    0 referências
    public class ProjectsController : ControllerBase//Clicar CTRL . > Enter
    {
        [HttpGet]//URL: api/projects?query=net core (url fica assim)
        0 referências
        public IActionResult Get(string query)//Adicionar uma query
        {
            // * FILTRAR OU BUSCAR TODOS
            return Ok();//Mesmo que enconte 0 elementos padrão retorna OK
        }
        // api/projects/3 ... ID do projeto
        [HttpGet("{id}")]// Adicionar o parametro ("{id}")
        //Retorno IActionResult > Mudar nome do metodo para GetById > (Parametro)
        1 referência
        public IActionResult GetById(int id)
        {
            // * LOCALIZAR O PROJETO
            // Return NotFound(); Caso não loclaize o projeto
            return Ok();//Retorna OK caso encontre o projeto
        }
        [HttpPost]
        0 referências
        public IActionResult Post([FromBody] CreateProjectModel createProject)
        {
            // Return BadRequest(); (Algum campo esta fora do padrão)
            if (createProject.Title.Length > 50)// Se Titulo tiver mais que 50
            {
                return BadRequest();
            }
            // * CADASTRAR O PROJETO
            // Ou voce coloca o nome para encontrar o objeto cadastrado ("GetById")
            // Ou usa: (nameof(GetById))
            //Usa 3 Parametros:            1 -   2 - Parametro paar URL detalhes3 - Objeto criado
            return CreatedAtAction(nameof(GetById), new { id = createProject.Id }, createProject);
        }
        // api/projects/2
        [HttpPut("{id}")]// Pode receber por URL ou corpo da requisição
        0 referências
        public IActionResult Put(int id, [FromBody] UpdateProjectModel updateProject)
        {
            if (updateProject.Description.Length > 200)
            {
                return BadRequest();
            }
            // * ATUALIZO OBJETO
            return NoContent();//Retorno padrão é NoContent
        }
        // api/projects/3 DELETE
        [HttpDelete("{id}")]
        0 referências
        public IActionResult Delete(int id)
        {
            // *BUSCAR SE NÃO EXISTIR RETORNAR: NotFound

            return NoContent();//Remover > Pode ser permanente não indicada, ou alterar uma propiredade
        }
    }
}
```



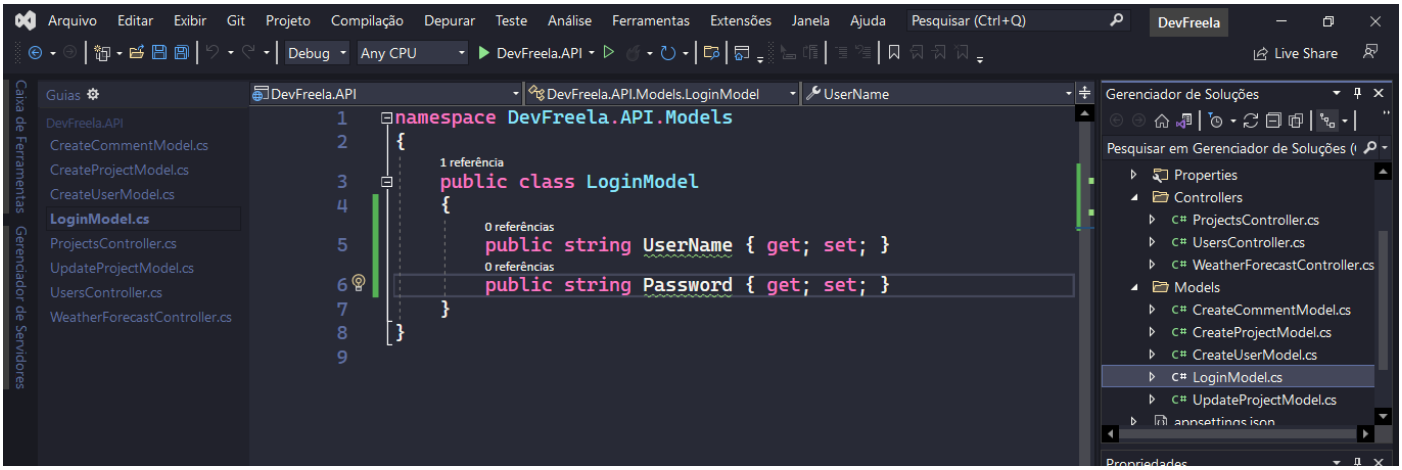
CRIANDO ENDPOINTS: Criar pontos de acessos; No caso de projetos já criamos os pontos de acesso junto com as Actions;

FUNCIONALIDADES IMPLEMENTADAS (A principio):
- Cadastro, Atualização, Cancelamento, Consulta de Projetos;
- Cadastro de Usuarios, Perfis Freelancer e Clientes;
- Adicionar comentários para um serviço Freelancer;
- Definir, Iniciar e Finalizar o Projeto;

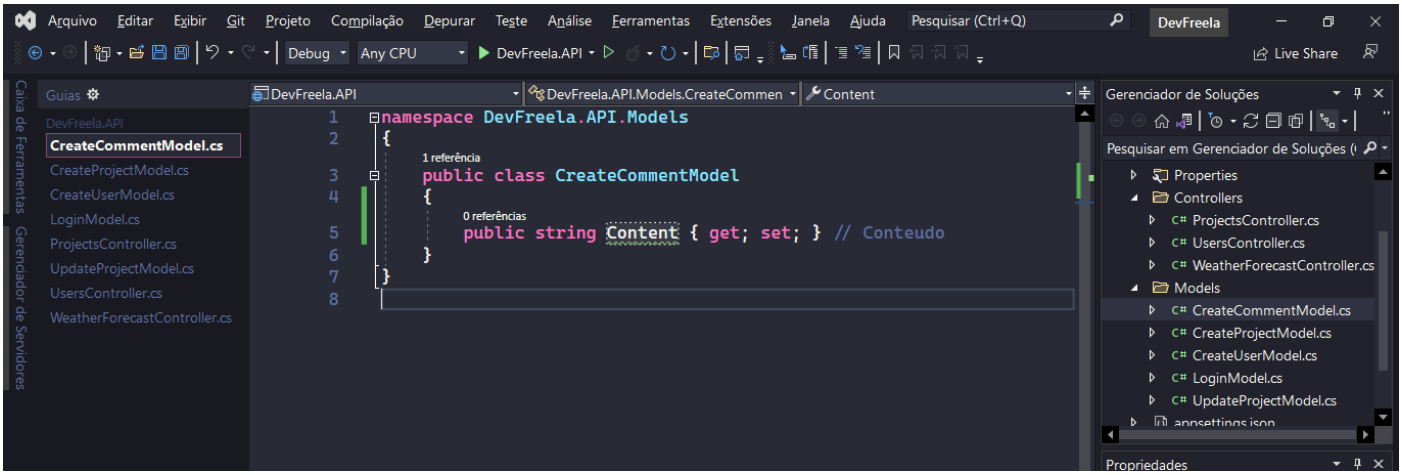
CADASTRAR PONTO DE ACESSO USUARIOS:
- Pasta Controller > Classe UsersController.cs



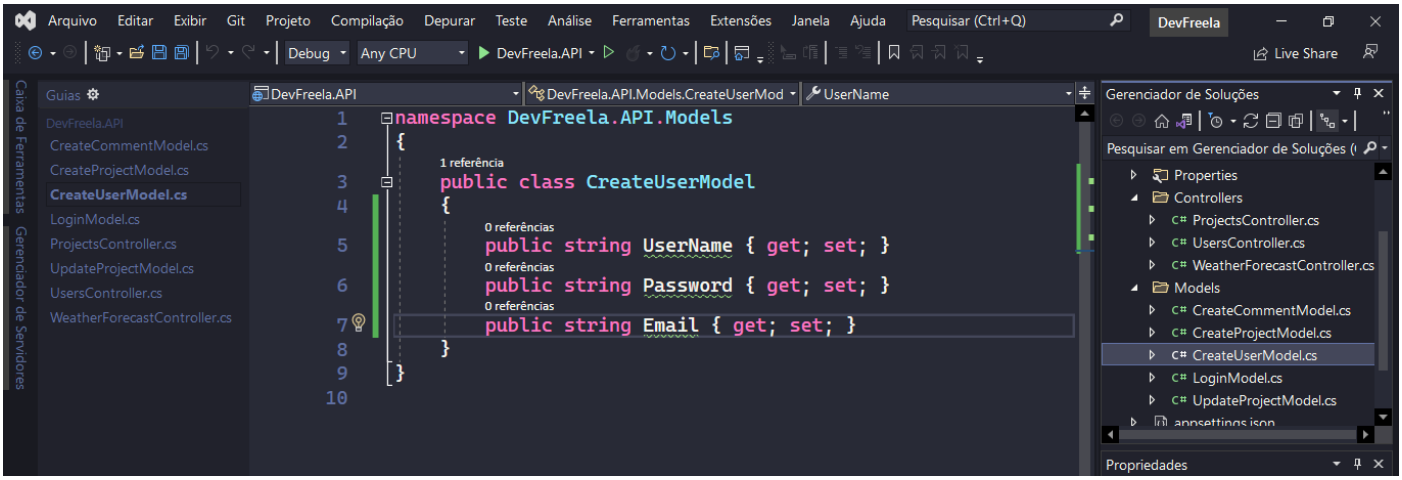
Pasta Models > LoginModel.cs



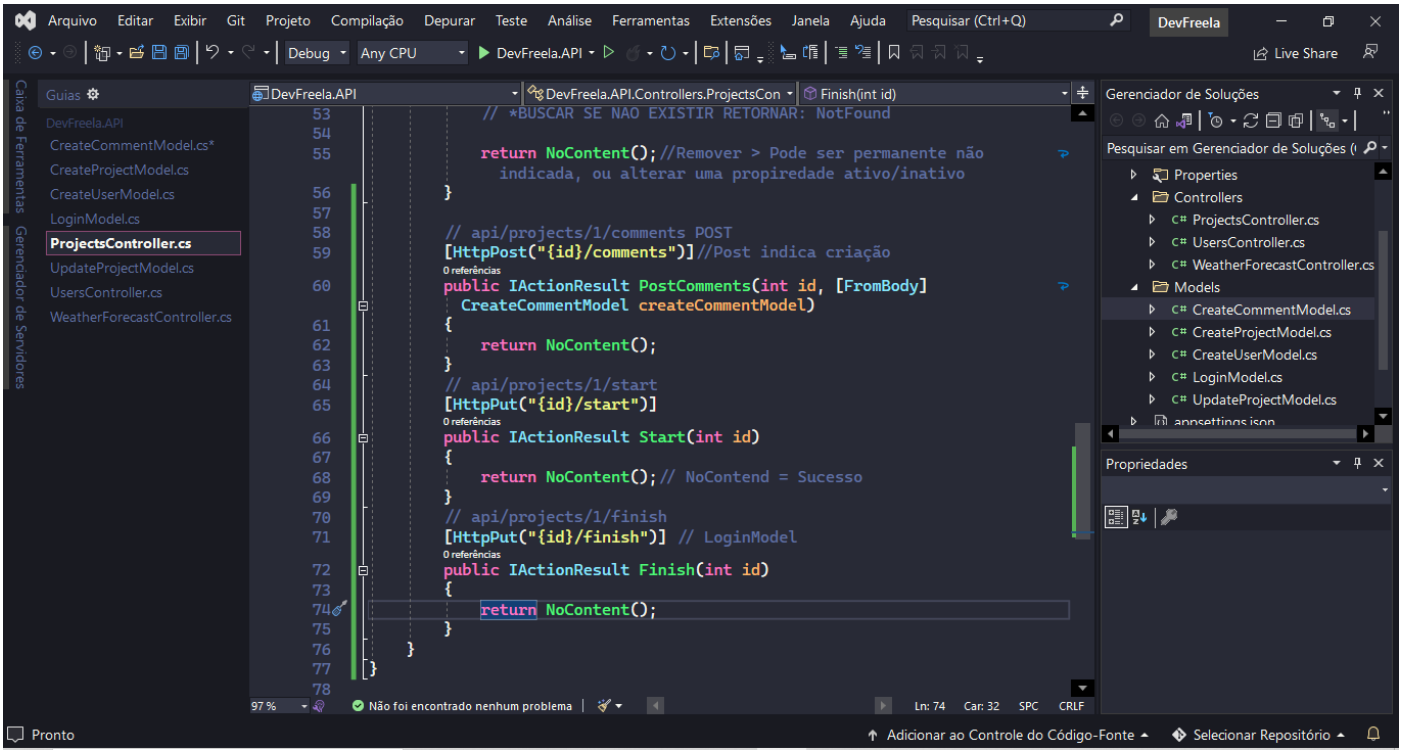
Pasta Model > CreateCommentModel



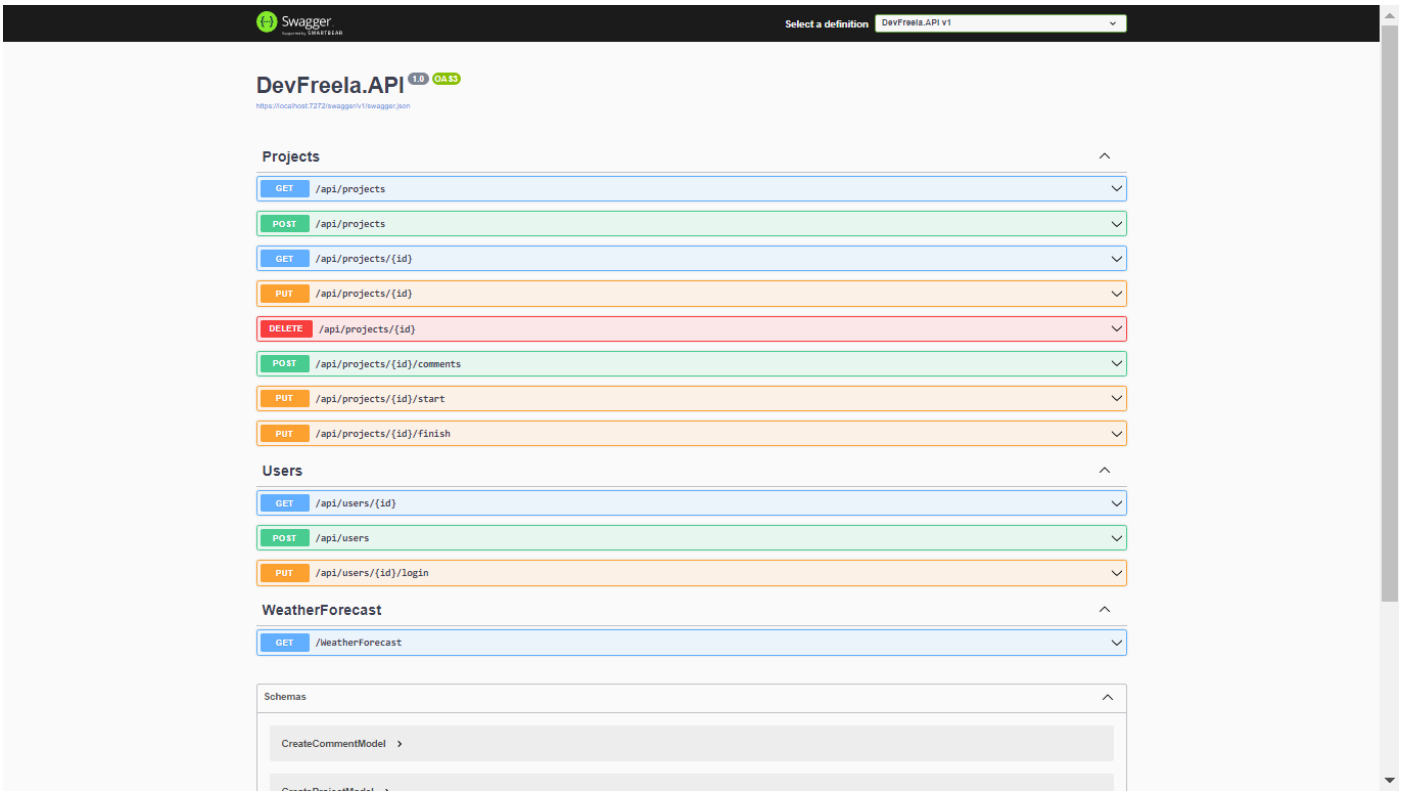
Pasta Model > CreateUserModel



ADIÇÃO EM PROJECTCONTROLLER



SWAGGER



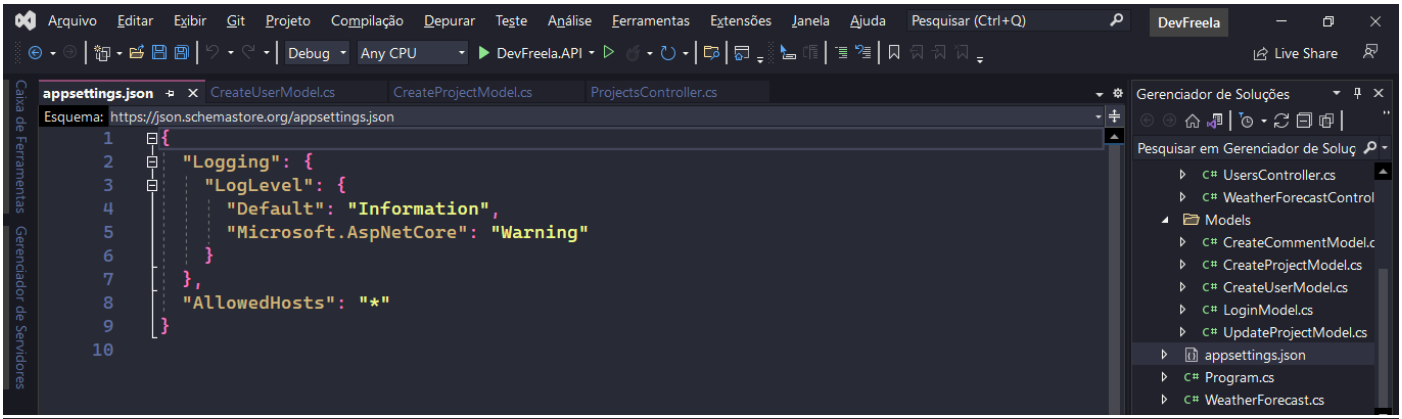
CONFIGURANDO A APLICAÇÃO

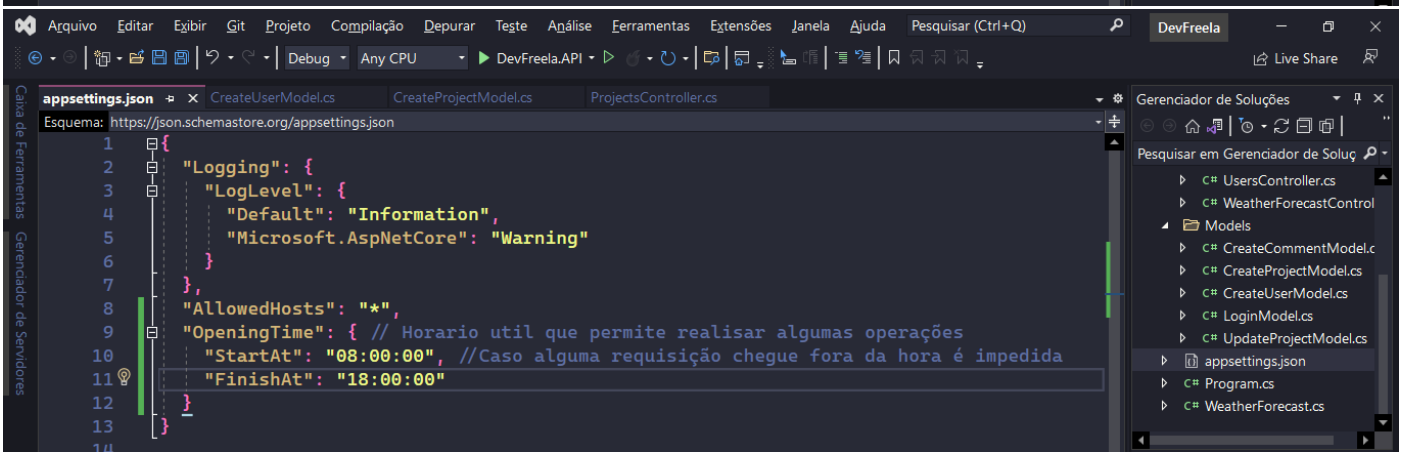
ASP .NET Core permite carregar configurações de uma serie de fontes, pela interface IConfiguration como:
- Arquivos de configuração, como appsettings.json (Criado assim que criamos um projeto)
- Variaveis de ambiente
- Argumentos de linha de comando
- Da nuvem Azure, como configurações de aplicativos, e pelo Azure Key Vault (Guarda segredos de api)

O arquivo de configuração podemos carregar varias versões dele, definido pela variável de ambiente APSNETCORE_ENVIRONMENT –EX:
- appsettings.Development.jason (Depende do ambiente)

- appsettings.Staging.json
- appsettings.Product

- A melhor maneira de obter configurações é usar o padrão Option; EX: appsettings.json (Pode ser usado como uma classe e receber configurações) - Queremos definir configurações para tempo de manutenção, de horário x a horário x. Pode adicionar uma serie de configurações:





CRIAR UMA CLASSE QUE REPRESENTE ESSA ESTRUTURA
Pasta Model > Classe OpeningTimeOption.cs

- Já no caso de linha de comando, basta passar como simples parâmetros: dotnet run nome=Luis e utilizar o código Configuration.GetValue<T>(“nome_parametro”)

INJEÇÃO DE DEPENDENCIA

- Técnica muito usada em projetos: Permite inserir uma dependencia de um Objeto através de um construtor;
- Metodo que usamos para adicionar configurações para o Controller, é um exemplo de injeção de dependência;
- Tecnica muito utilizada para melhorar a qualidade do código em refatorações e torna-lo mais testável;
- Retira dependências internas do código para o construtor;
- Se não usar ela em muitos contextos o teste unitário se torna bem difícil;

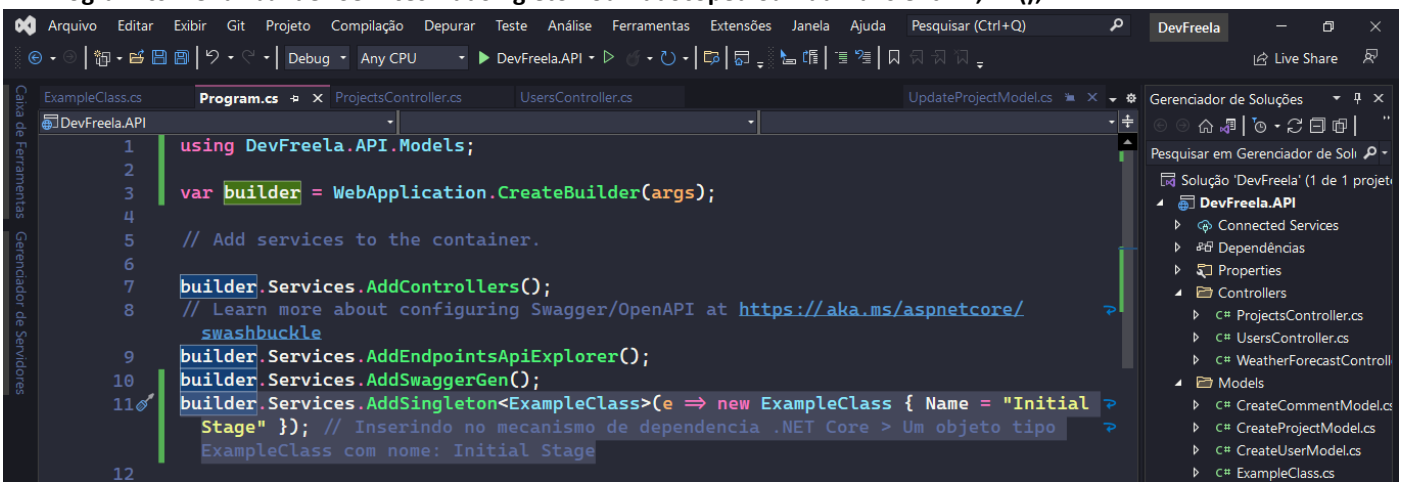
Cada um deles define um ciclos de vida do Objeto:

- Singleton: Uma instancia por aplicação; Todas as requisições vão receber o mesmo Objeto, se permitir que seu código altere o código Singleton altera tudo, não desejável;
- Scoped: Uma instancia por requisição; EX: Recebe uma requisição no Controller e so altera a classe que tiver herdando uma da outra;
- Transient: Uma instancia por classe; Ao delegar o Controller para outra classe, a primeira não mais a mesma;

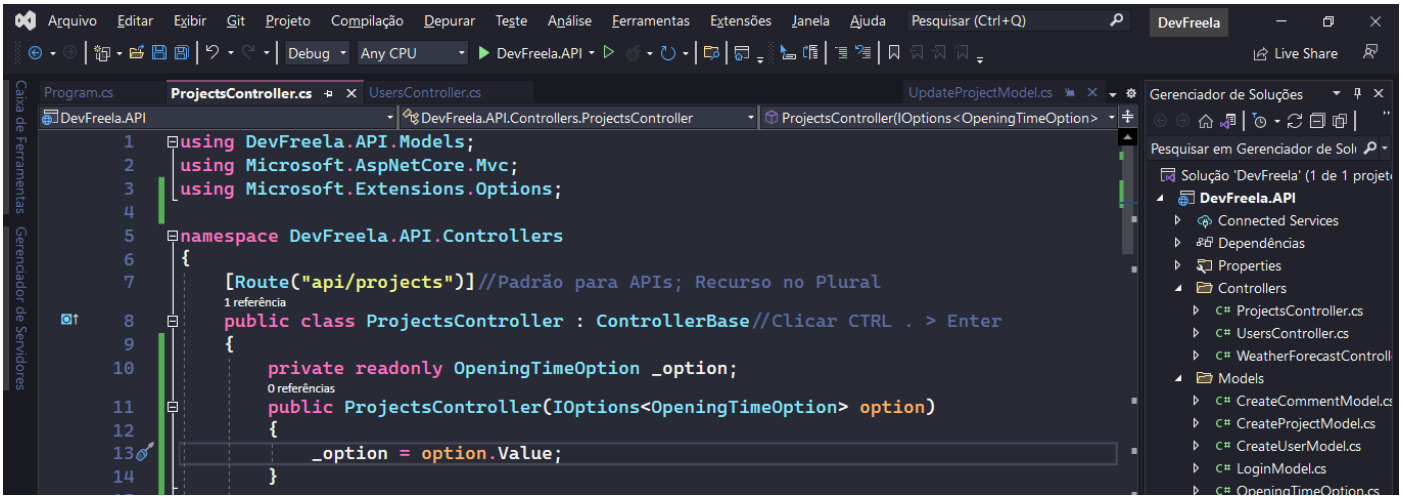
Uma observação aqui para facilitar sua progressão no curso.
Com a versão mais recente sendo o .NET 6, talvez seja confuso essa parte de Injeção de Dependência que está por vir, mas venho te tranquilizar que a mudança é simples.

No .NET 6, a classe Program vai ser responsável por essa configuração. Mas vejo como está simples.
Antes: Método ConfigureServices(IServiceCollection services) na classe Startup, onde é configurada a injeção de dependência
Agora: No arquivo Program.cs, basta acessar a propriedade da variável builder.Services para realizar a operação que era feita com o services (de tipo IServiceCollection).
Ou seja, ao invés de
services.AddSwaggerGen(.....
services.AddScoped<IA, A>();
Está
builder.Services.AddSwaggerGen(.....
builder.Services.AddScoped<IA, A>();
A mesma lógica se aplica ao restante de métodos de configuração, como o do DbContext (quando chegar no Entity Framework Core vai ficar claro), MediatR, FluentValidations, etc.

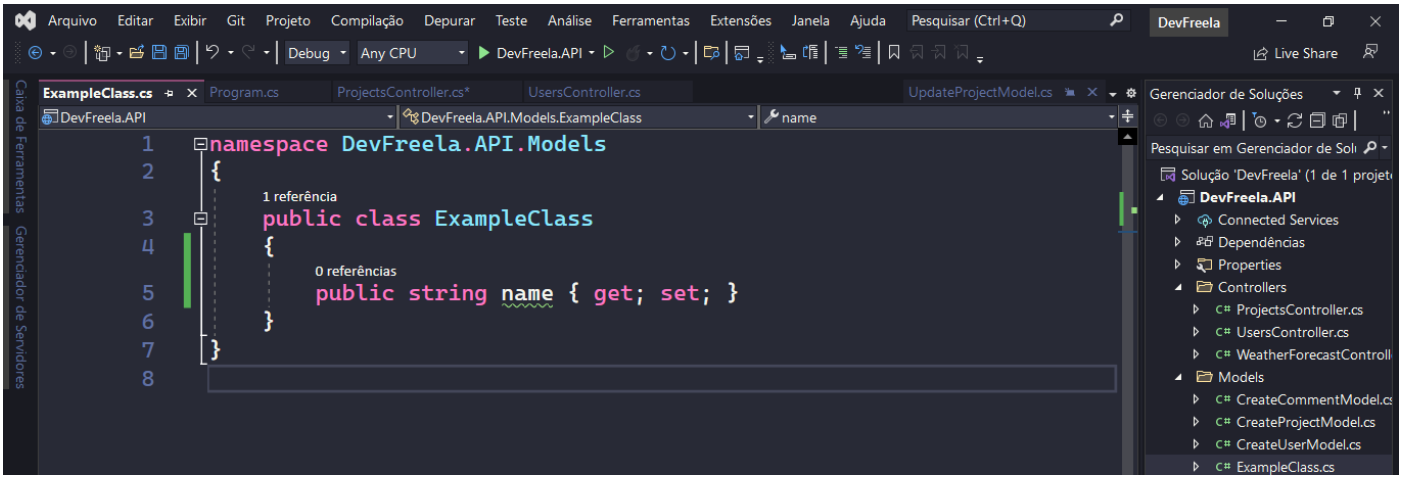
Em Program.cs > Criar builder.Services.AddSingleton ou AddScoped ou AddTransient<IA, A>();



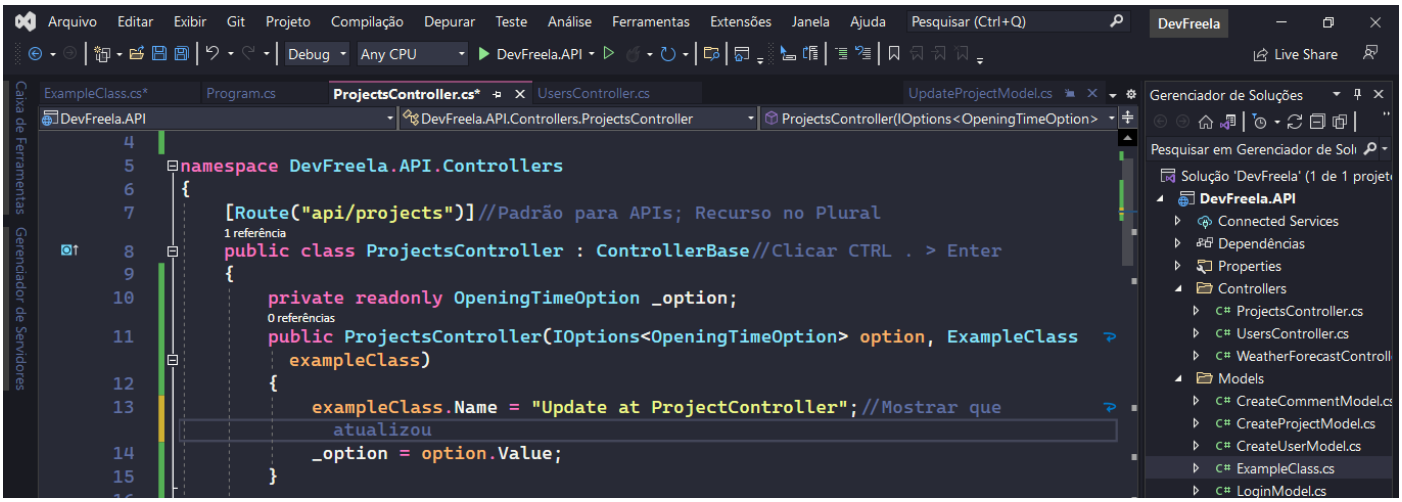
Em ProjectController > Criar o Options (Injeção de dependência)



MODELS > Criar ExampleClass.cs (Mostrando como funciona o ciclo de vida de um Objeto)



ALTERAR ProjectsController > Para receber via injeção de dependência



Adicionar a dependência em UserController