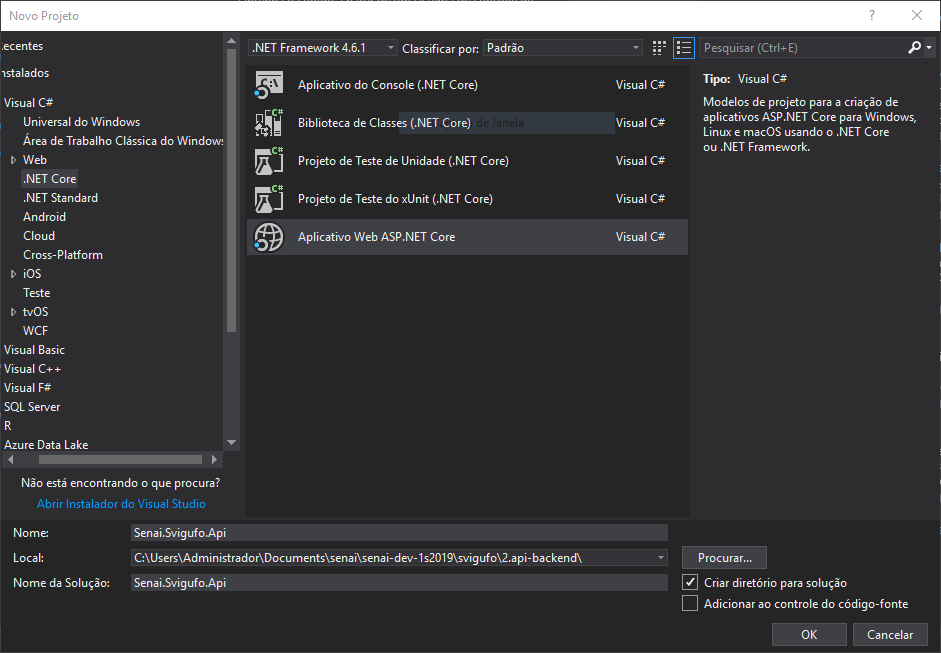
1. API
2. Startup – configurar o MVC e a versão que eu estou utilizando
3. TipoEventoController
4. TipoEventoDomain
5. ITipoEventoRepository
6. TipoEventoRepository
7. InstituicaoController
8. InstituicaoDomain
9. IInstituicaoRepository
10. InstituicaoRepository
11. Validar dados de entrada com anotação
12. UsuarioDomain
13. UsuariosController
14. IUsuarioRepository
15. UsuarioRepository

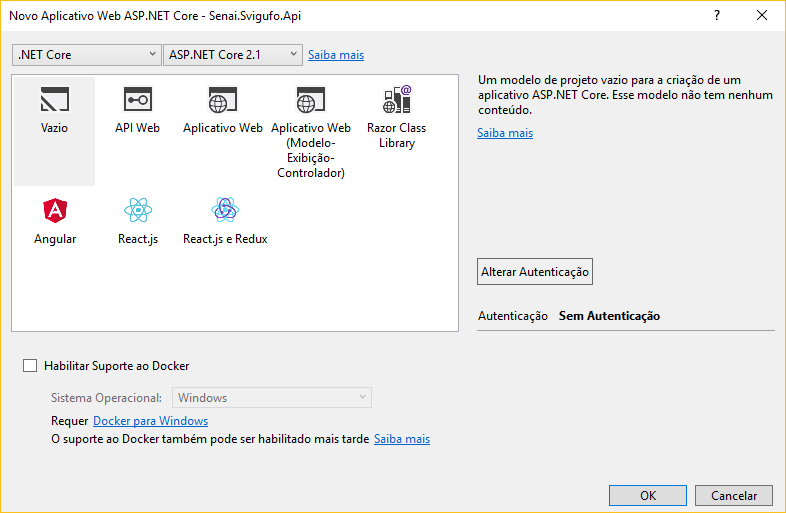
Explicar todo o contexto de API com imagens e apresentações.

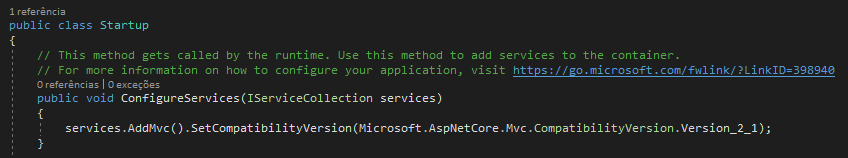
Explicar o que é JSON, API, vantagens

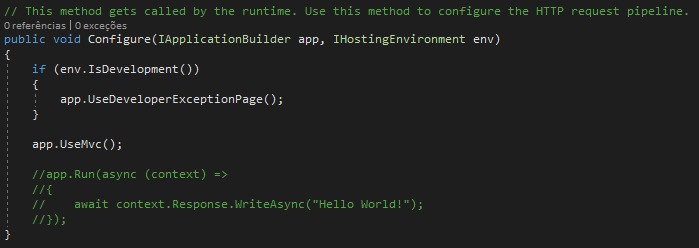
Somente após explicar todo o contexto, mostrar um projeto sendo criado

Criar um novo projeto WebApi

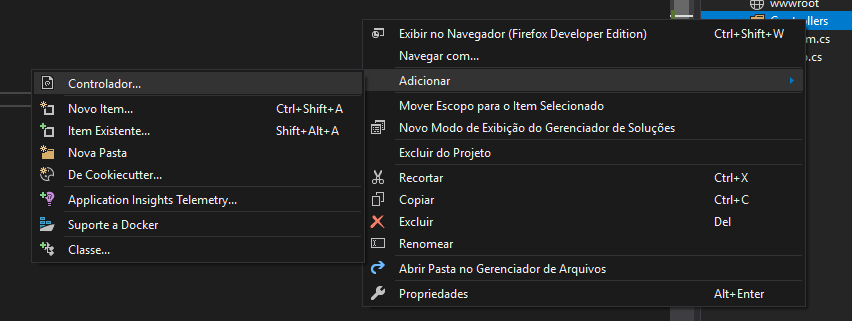






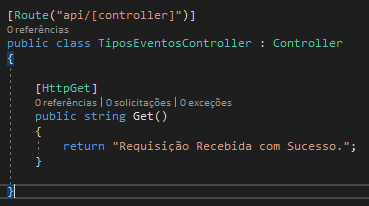


Nova pasta Controllers



QUANDO CRIAR O CONTROLLER, LEMBRAR DE FALAR SOBRE O CONTROLLERBASE – QUE ATUA SEM O SUPORTE A VIEW, POIS NÃO IREMOS PRECISAR DELA

TiposEventosController

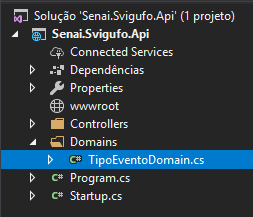


Realizar requisição

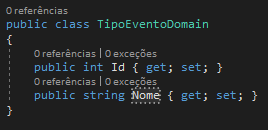
Mostrar na web – Navegador, a informação recebida – Não iniciar com o postman ainda

Criar nova pasta – Domains

Criar nova classe TipoEventoDomain



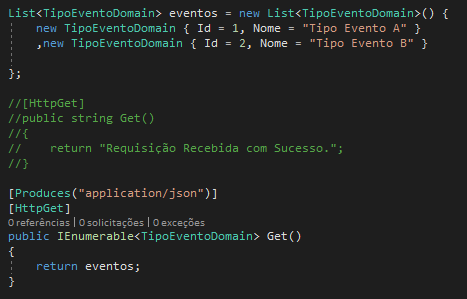
Incluir as propriedades



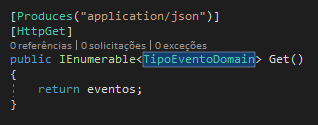
<https://stackoverflow.com/questions/23196931/returning-ihttpactionresult-vs-ienumerableitem-vs-iqueryableitem>

Diferença entre IEnumerable e IActionResult

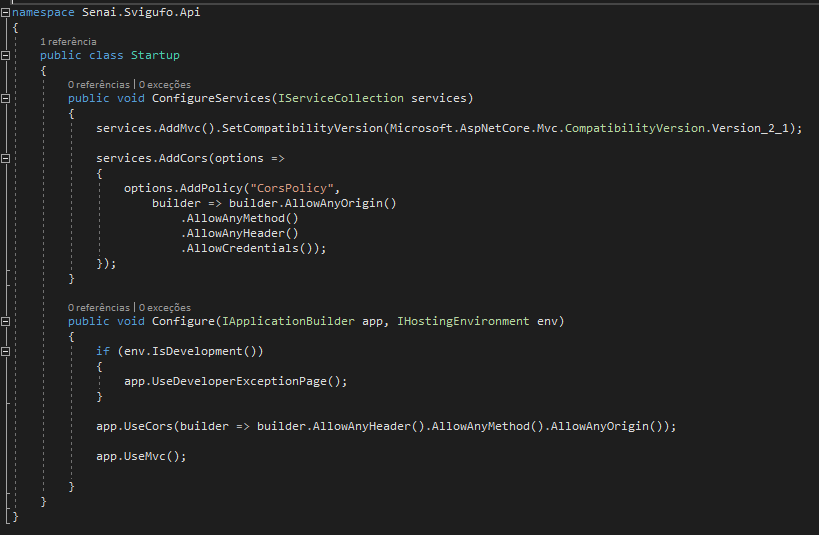
Realizar um retorno simples com uma lista local



Realizar a requisição



LEMBRAR DE INCLUIR O CORS NA NOSSA APLICAÇÃO PARA MOSTRAR O FRONT



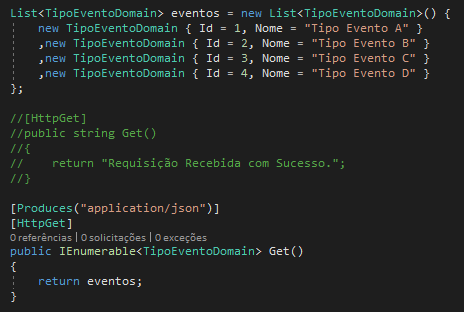
Mostrar o front-end fazendo a requisição para o back-end

E mostrar que sua implementação não é conhecida

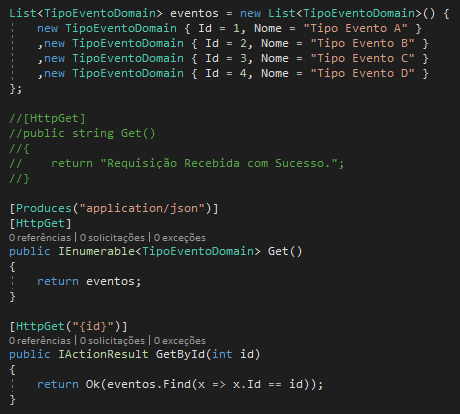
Eu nem mesmo sei/conheço aonde esses dados estão sendo armazenados

Adicionar mais dois eventos e mostrar o resultado

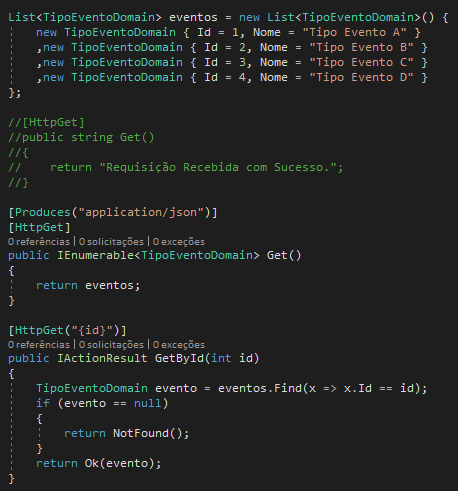
E também mostrar a interação no front-end

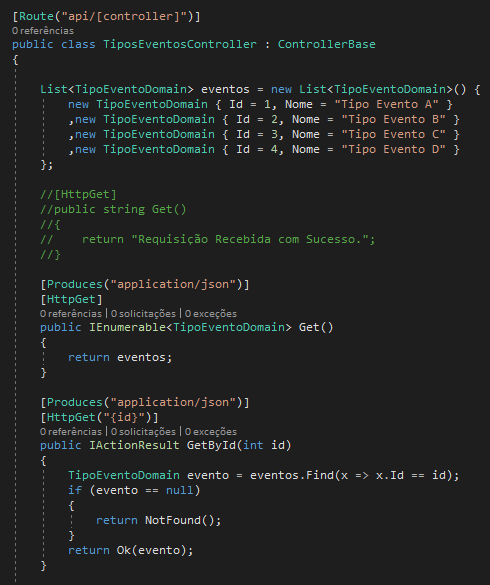


Buscando somente um



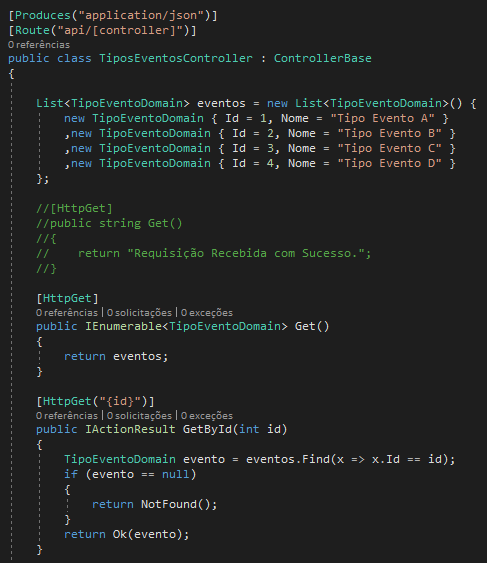
Melhorar realizando um if para ver se o evento foi encontrado





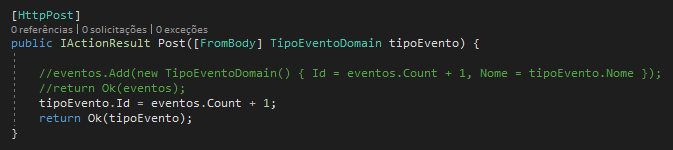
A saída também será produces

Podendo assim, colocar no controller diretamente

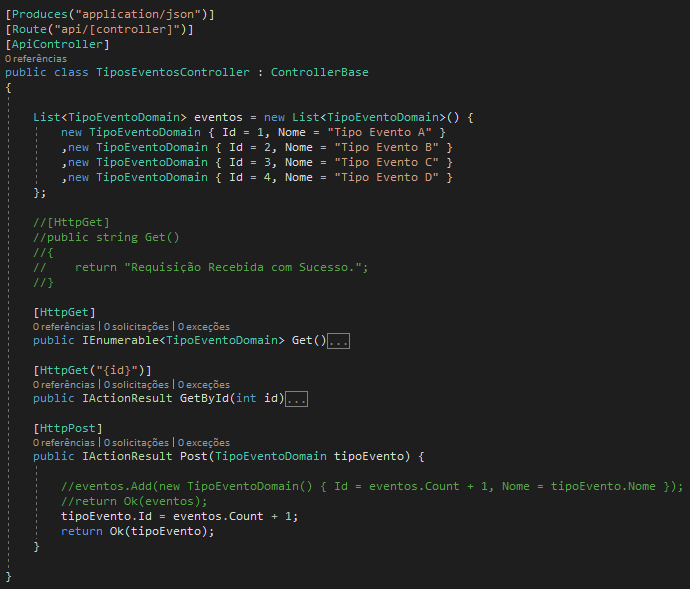


Trabalhando com POST

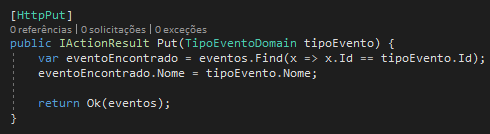
MOSTRAR SOMENTE AQUI O POSTMAN



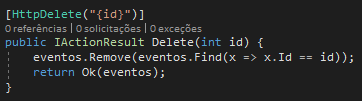
Explicar a anotação [ApiController]



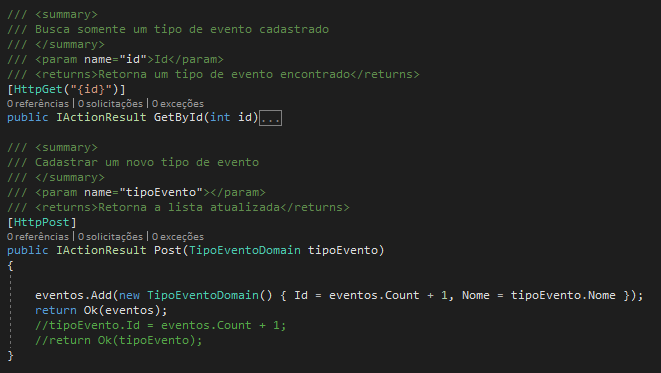
Mostrar o PUT



DELETE

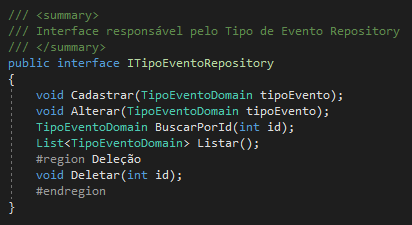


INCLUIR COMENTÁRIOS NO CÓDIGO

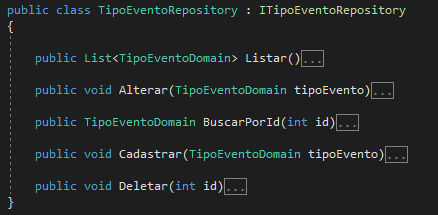


BANCO DE DADOS

Criar uma nova interface

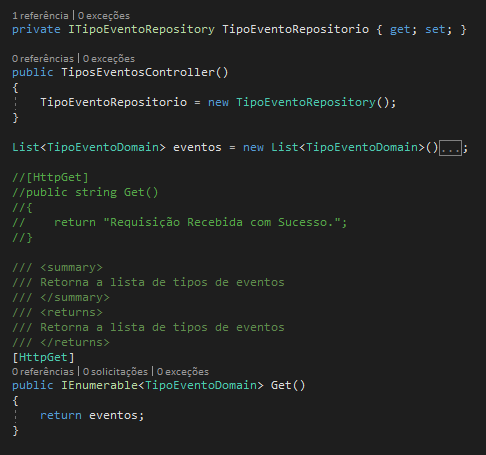


Criar um novo repositório

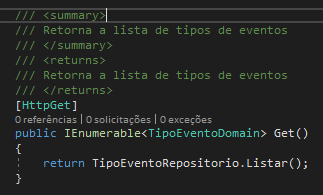


No controller, vamos alterar para ao invés de trabalhar com a lista local, chamarmos do nosso banco de dados.

Cria o construtor no controller

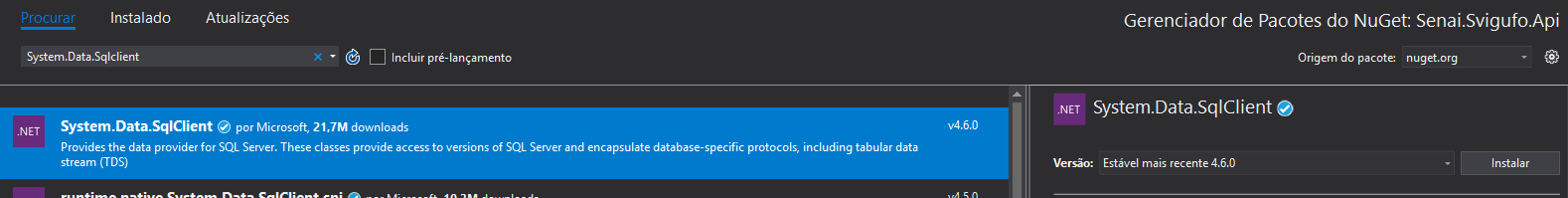


Todo a sua implementação continua a mesma



Porém, os dados virão agora do banco de dados

Botão direito -> gerenciar pacotes do nuget



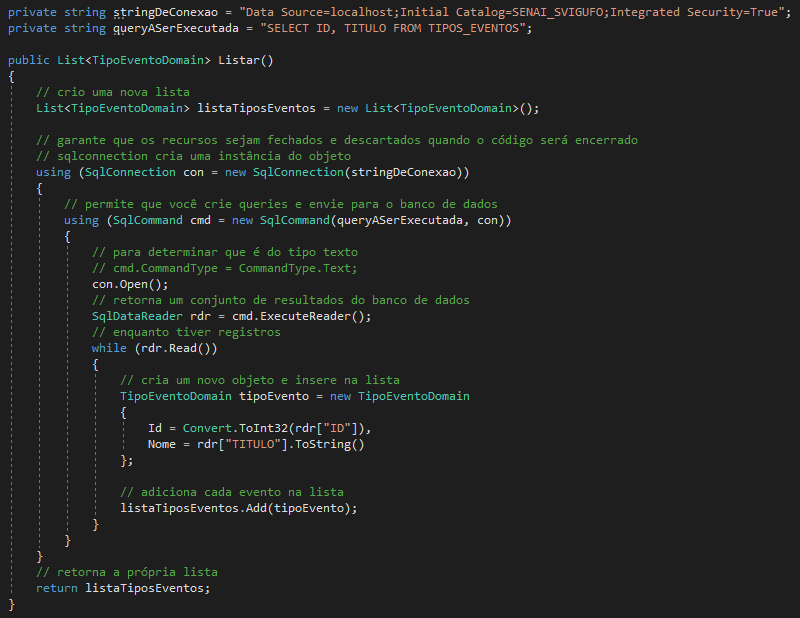
ADO.NET

<https://docs.microsoft.com/pt-br/dotnet/framework/data/adonet/>

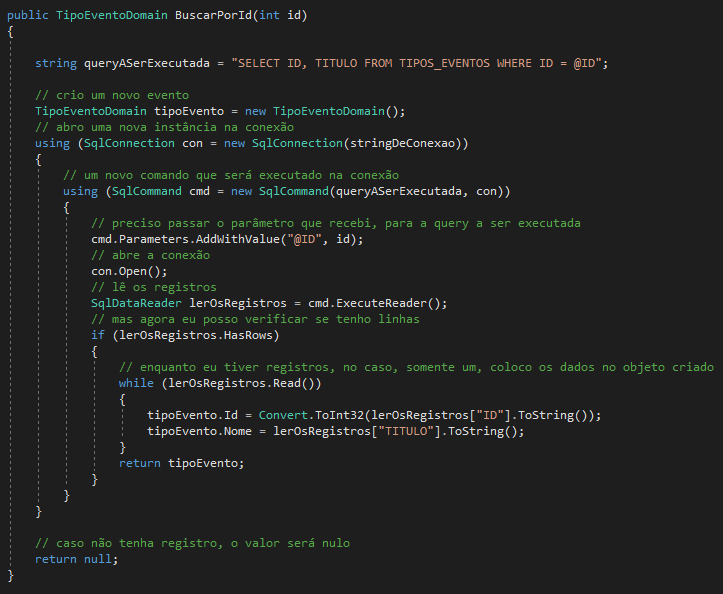
* ExecuteScalar is typically used when your query returns a single value. If it returns more, then the result is the first column of the first row. An example might be SELECT @@IDENTITY AS 'Identity'.
* ExecuteReader is used for any result set with multiple rows/columns (e.g., SELECT col1, col2 from sometable).
* ExecuteNonQuery is typically used for SQL statements without results (e.g., UPDATE, INSERT, etc.).

Implementação do Repositório

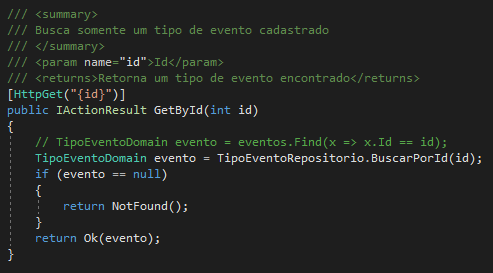
LISTAGEM



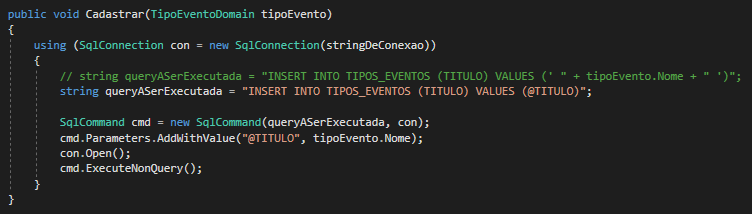
BUSCAR SOMENTE UM



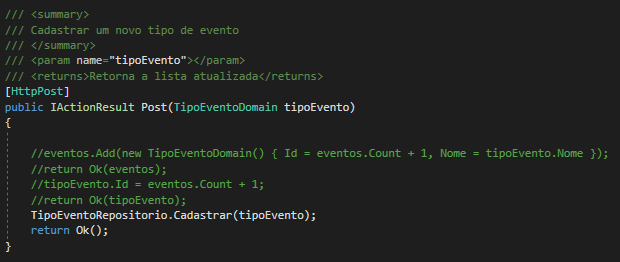
Mudo a implementação no controller, para realizar a busca agora no repositório



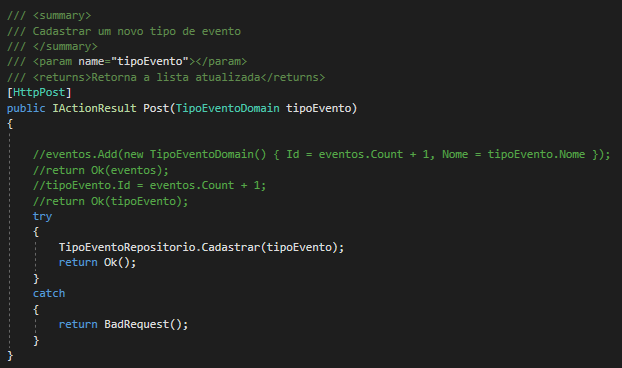
Alterar o repositório para cadastrar um novo tipo de evento



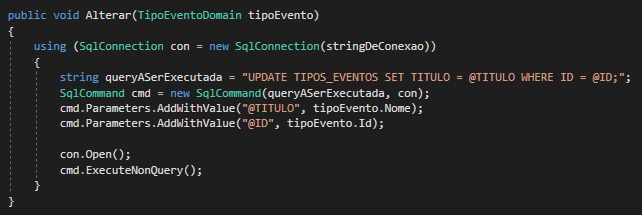
Alterar o controller



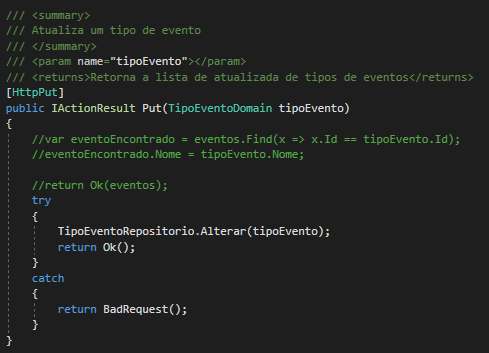
Realizar uma verificação na inserção para não travar a inserção



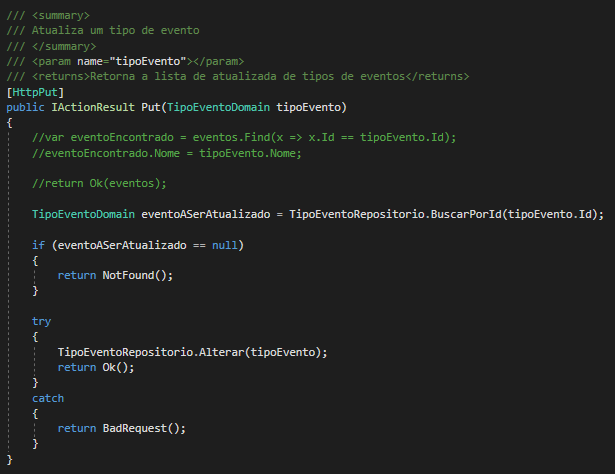
Alterar a implementação no repositório para atualizar



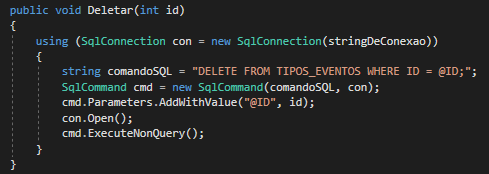
Ajustar a implementação no controller



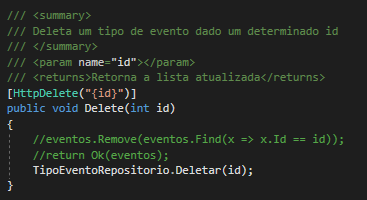
E caso o meu tipo de evento não seja encontrado, posso retornar um valor não encontrado.



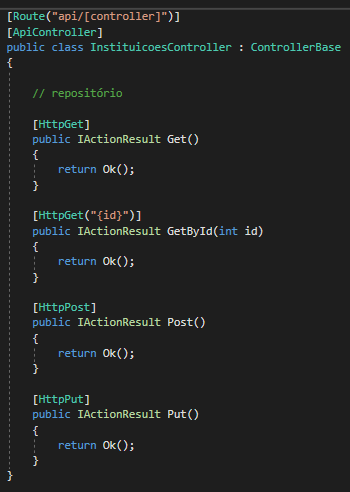
Alterar o repositório para deletar um registro



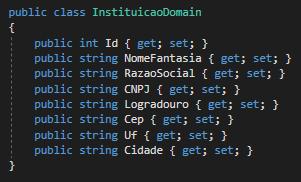
Deletar um tipo de evento



InstituicoesController

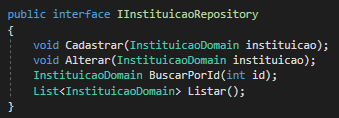


Adicionar um novo domínio

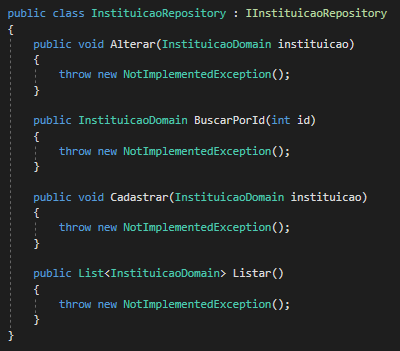


Olhar todas as propriedades do banco de dados.

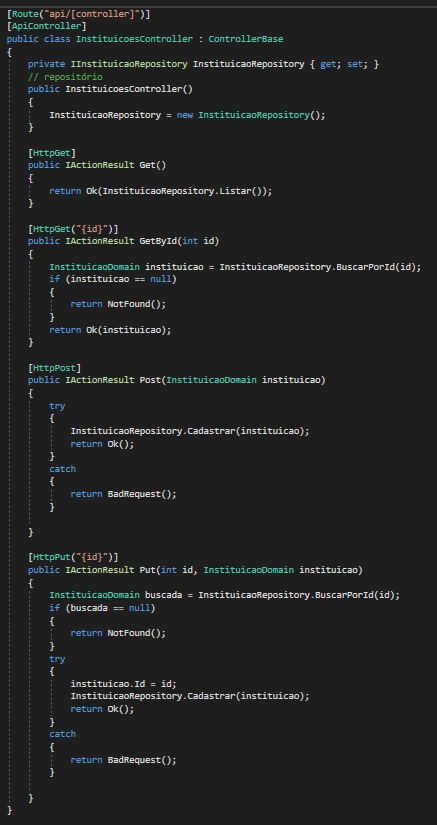
Criar interface



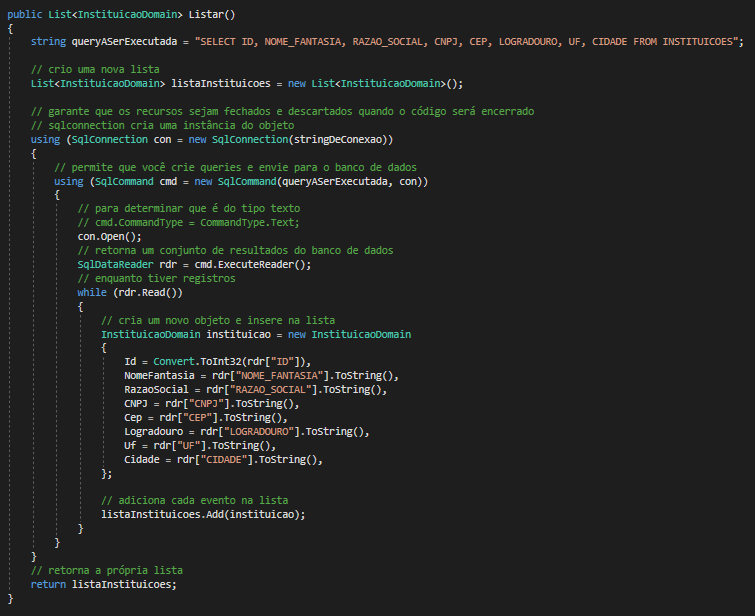
Criar repositório

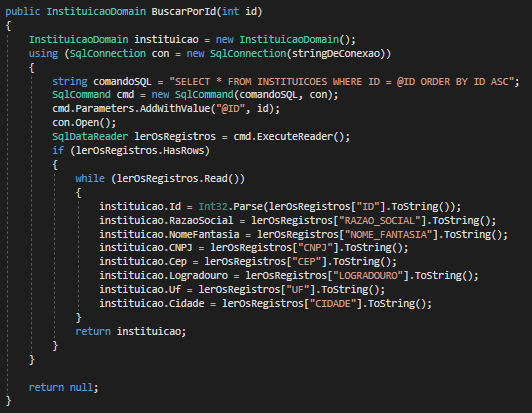


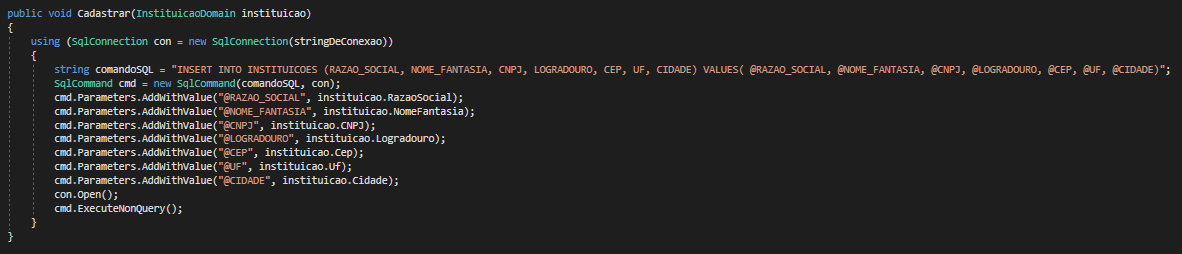
Ajustar os controllers

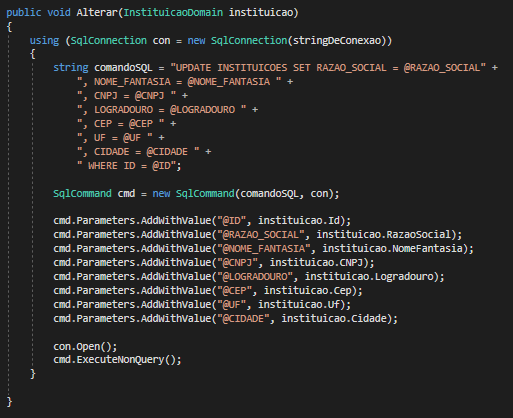


Ajustar os repositórios







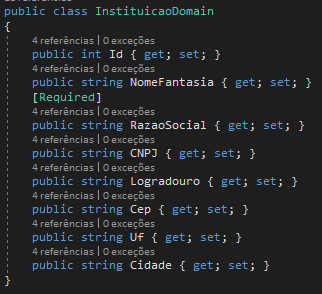


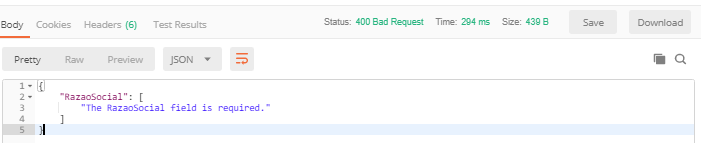
ENDPOINTS

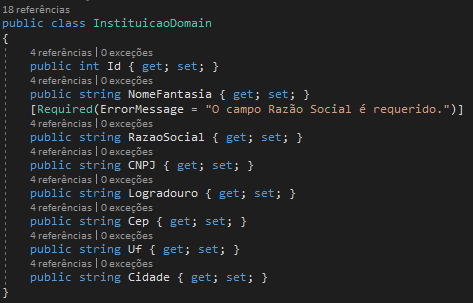
DOCUMENTAÇÃO

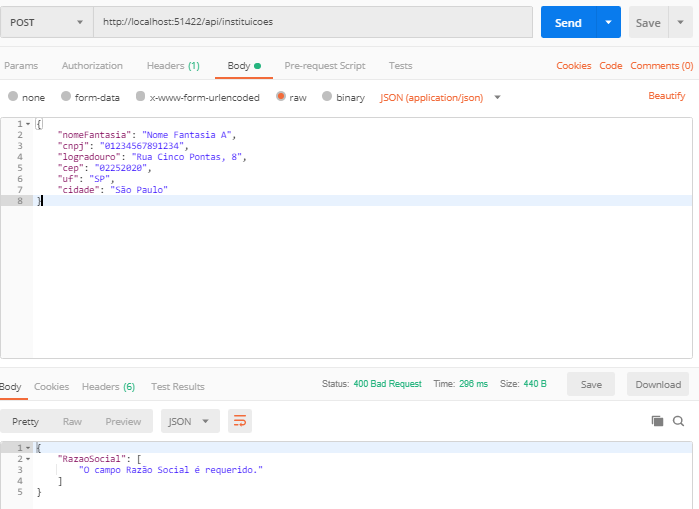
https://docs.microsoft.com/pt-br/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-2.2&tabs=visual-studio

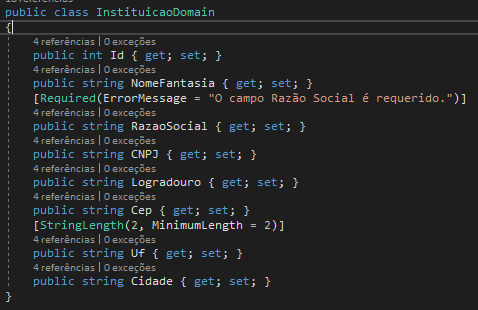
VALIDAR DADOS DE ENTRADA COM ANOTAÇÕES

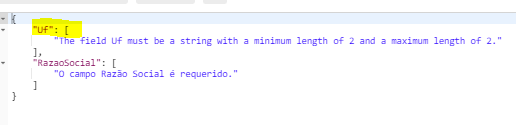


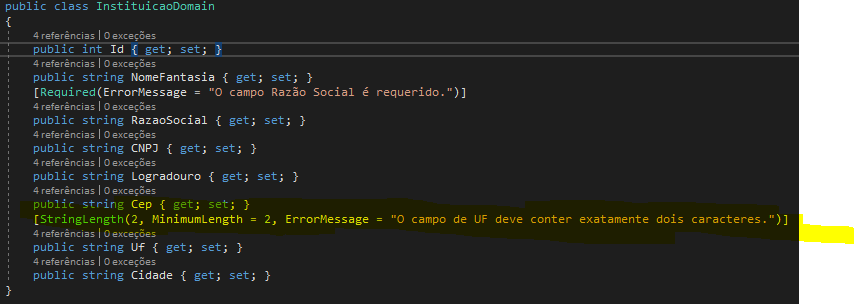


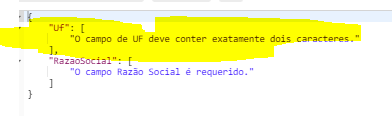




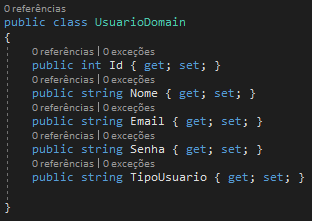


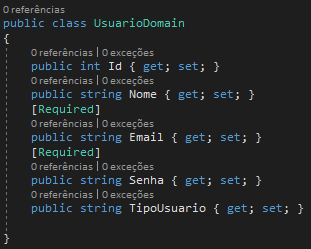






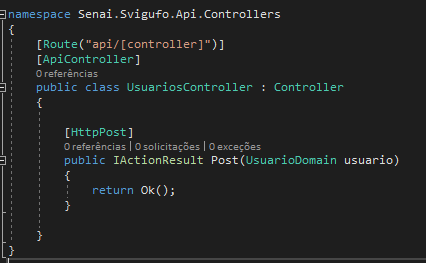
Criando um novo UsuarioDomain



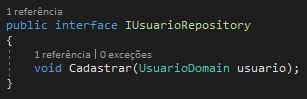


UsuariosController

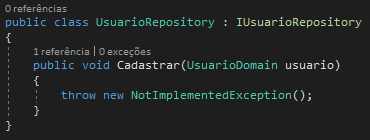
Vamos apenas cadastrar um novo usuário para depois fazermos o login



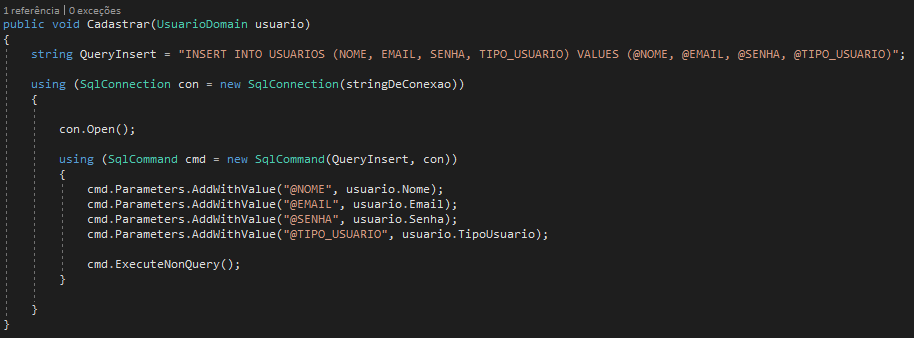
Criar a Interface para cadastrar



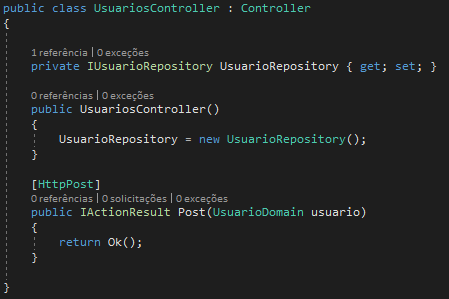
Criar o repositório para acesso ao banco de dados

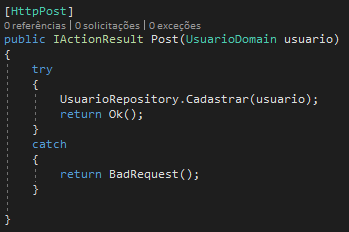


Incluir a implementação no repositório



Incluir no construtor o repositório

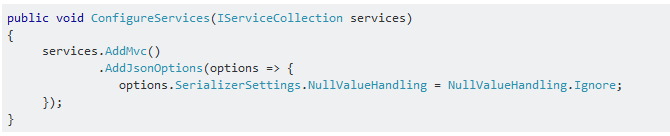




EXTRAS

IGNORAR OS NULOS NA SAÍDA DO JSON

Startup.cs



INJEÇÃO DE DEPENDÊNCIA

