Dado um determinado problema para o desenvolvimento, vamos criar o banco de dados e o seu respectivo projeto de BackEnd.

**Banco de Dados**

Definição das tabelas e sua estrutura.

01\_peoples\_ddl.sql

Inserção dos registros solicitados.

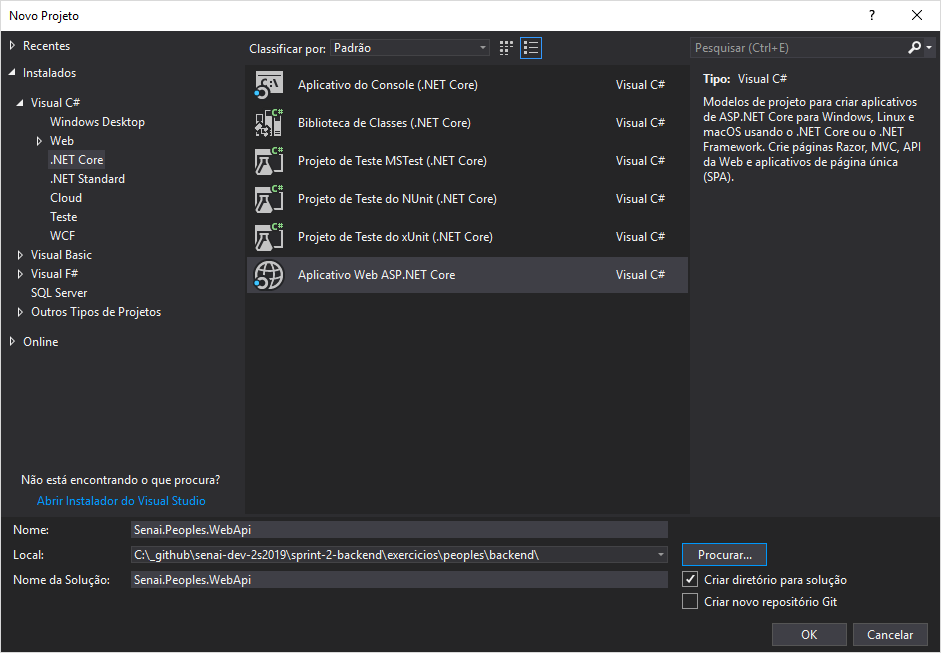
02\_peoples\_dml.sql

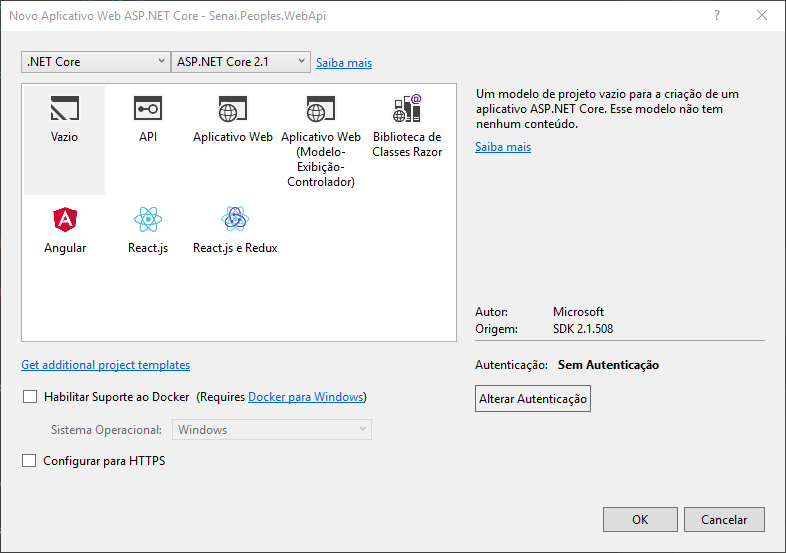
Seleção dos registros mencionados.

03\_peoples\_dql.sql

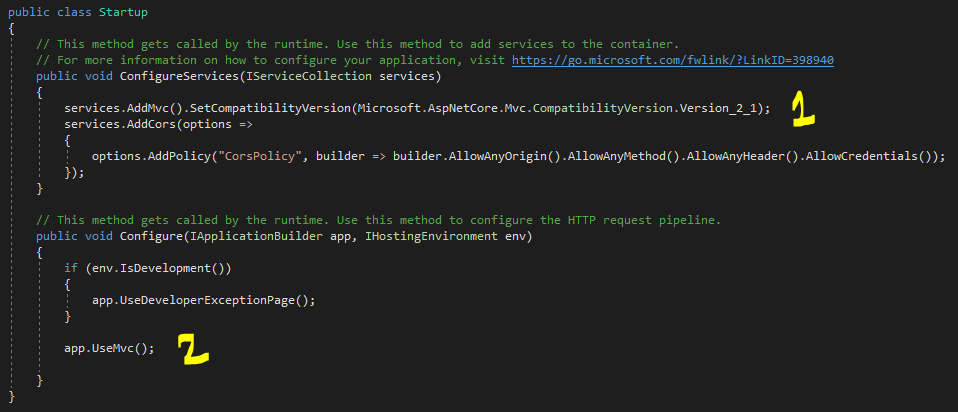
**BackEnd**

Criar um novo projeto de BackEnd chamado Senai.Peoples.WebApi (Nome da Empresa. Nome do Projeto. Tipo do Projeto).



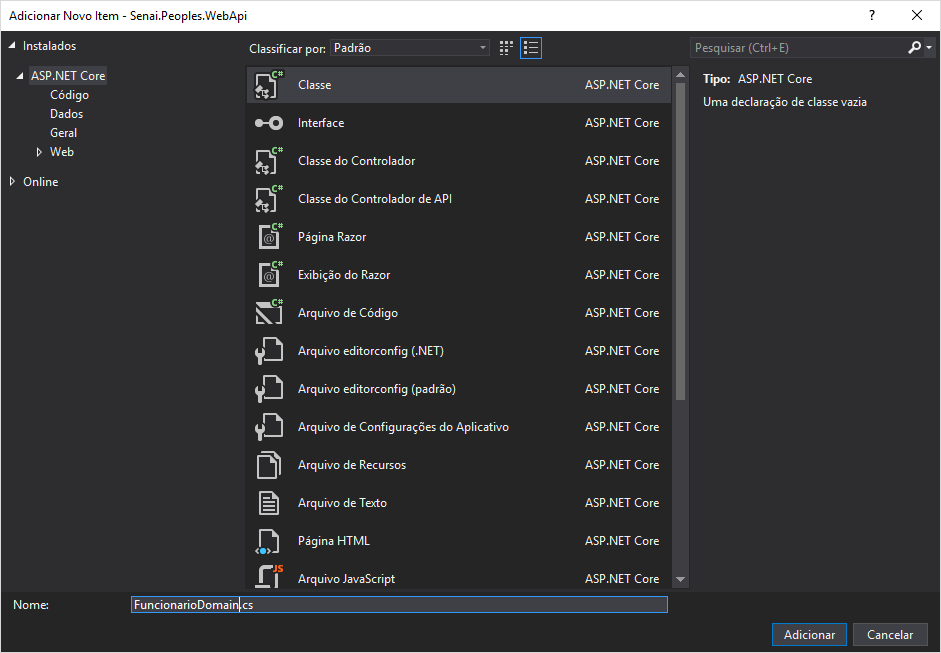


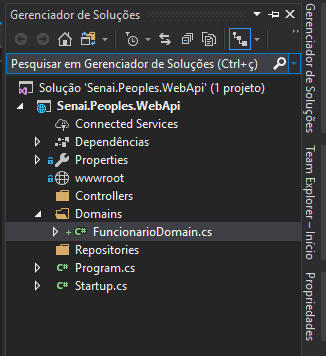
Configurar o Startup.cs



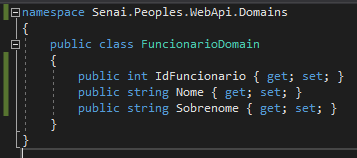
Como as entidades já foram definidas na base de dados, nós podemos começar com a criação dos nossos modelos em nossa aplicação. Ou domínios.

Criar uma pasta chamada *Domains* e criar um novo item chamado *FuncionarioDomain*.

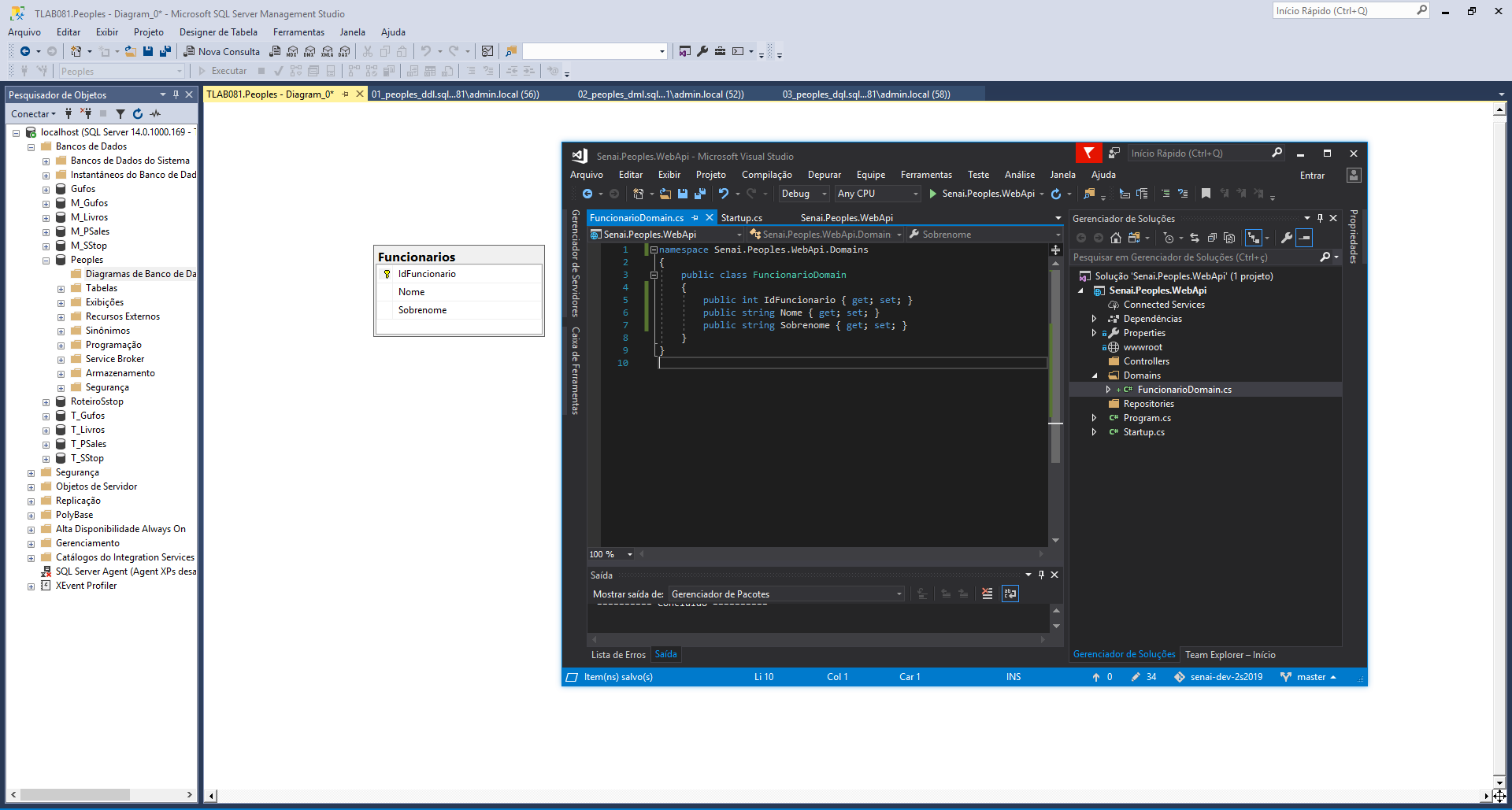




Uma vez que a estrutura foi definida na base de dados, podemos incluir as mesmas propriedades que foram definidas na tabela, no domínio da nossa aplicação.

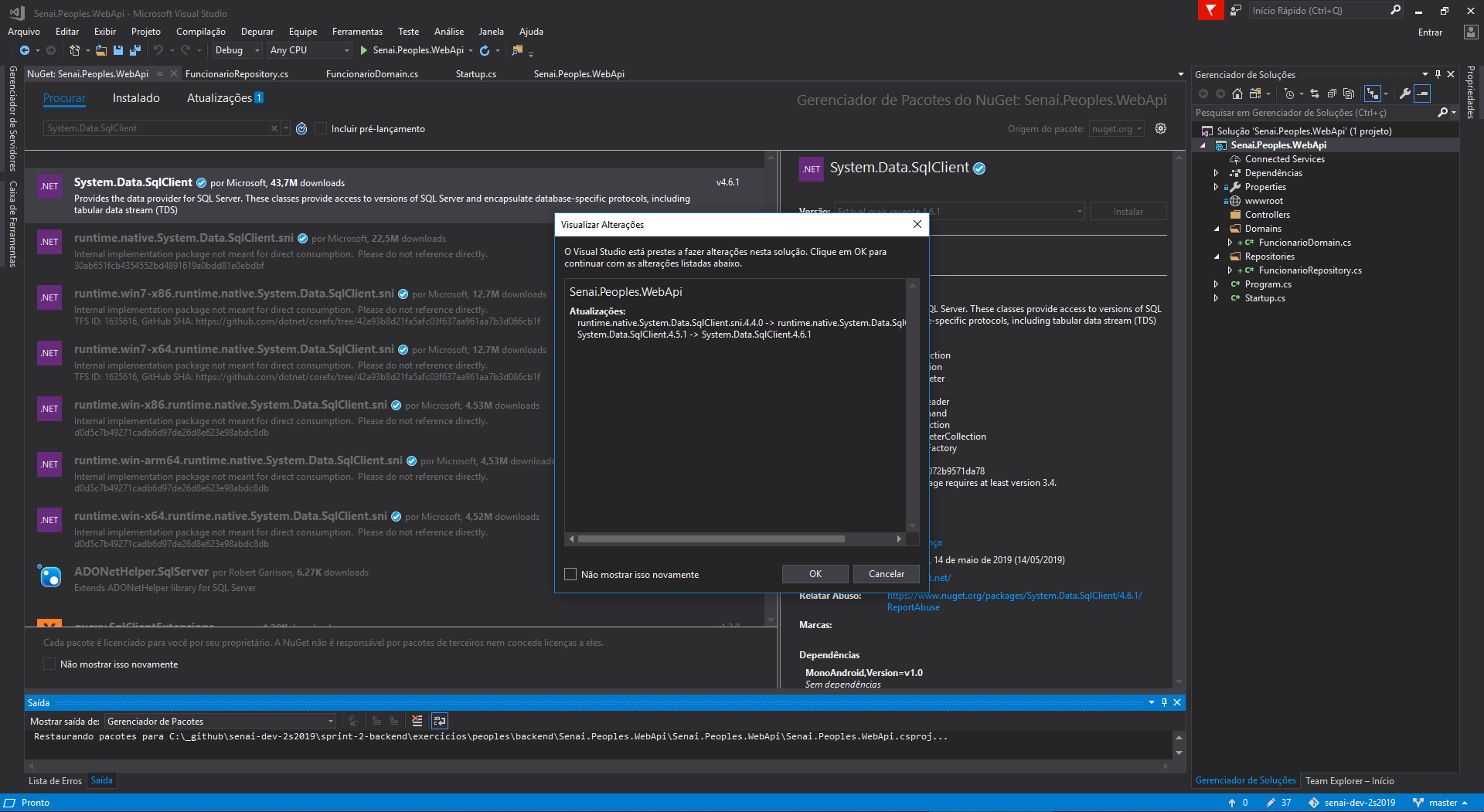


Se verificarmos o banco com a nossa aplicação, verificamos que eles mantêm as mesmas características.

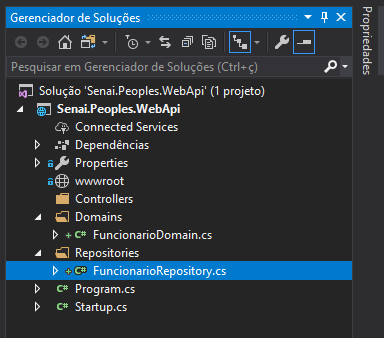


Antes de criar o controller correspondente para a ação, preciso ter o acesso ao banco para trazer os registros do banco de dados. Dentro da nossa estrutura, estamos criando por camadas o nosso projeto.

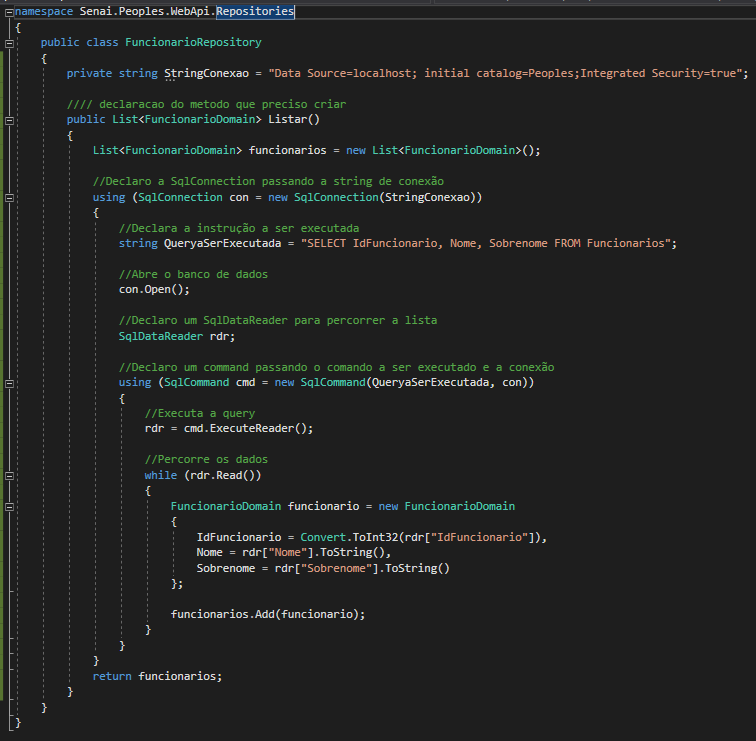
Sendo assim, além de criar uma classe chamada FuncionarioRepository.cs, preciso adicionar o SqlClient em nosso projeto para que eu tenha acesso ao banco de dados (consiga realizar a conexão e realizar todas as ações que desejo).



Criar a classe.



Vamos criar o primeiro método de listar todos os itens do banco de dados.



O que é importante na declaração do método é:

Qual é a ação que eu desejo realizar? Listar.

Listar();

O que eu desejo receber no retorno? Uma lista de funcionários.

List<FuncionarioDomain> Listar();

Qual a visibilidade? Pública.

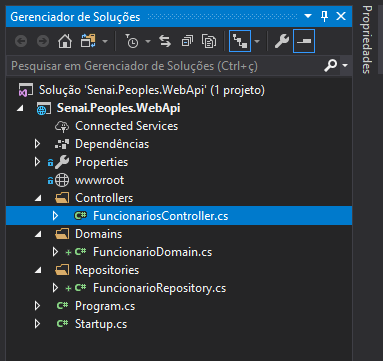
public List<FuncionarioDomain> Listar();

No método, há a descrição de cada comando a ser executado.

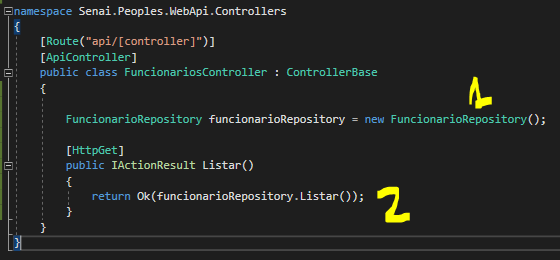
O que é importante do repositório é fazer o acesso aos dados. Neste caso, ele está fazendo a busca no banco de dados e retornando as informações que lá foram encontradas.

Adicionar um novo controlador que será responsável por nossas requisições.

Criar um controlador API chamado FuncionariosController.cs.



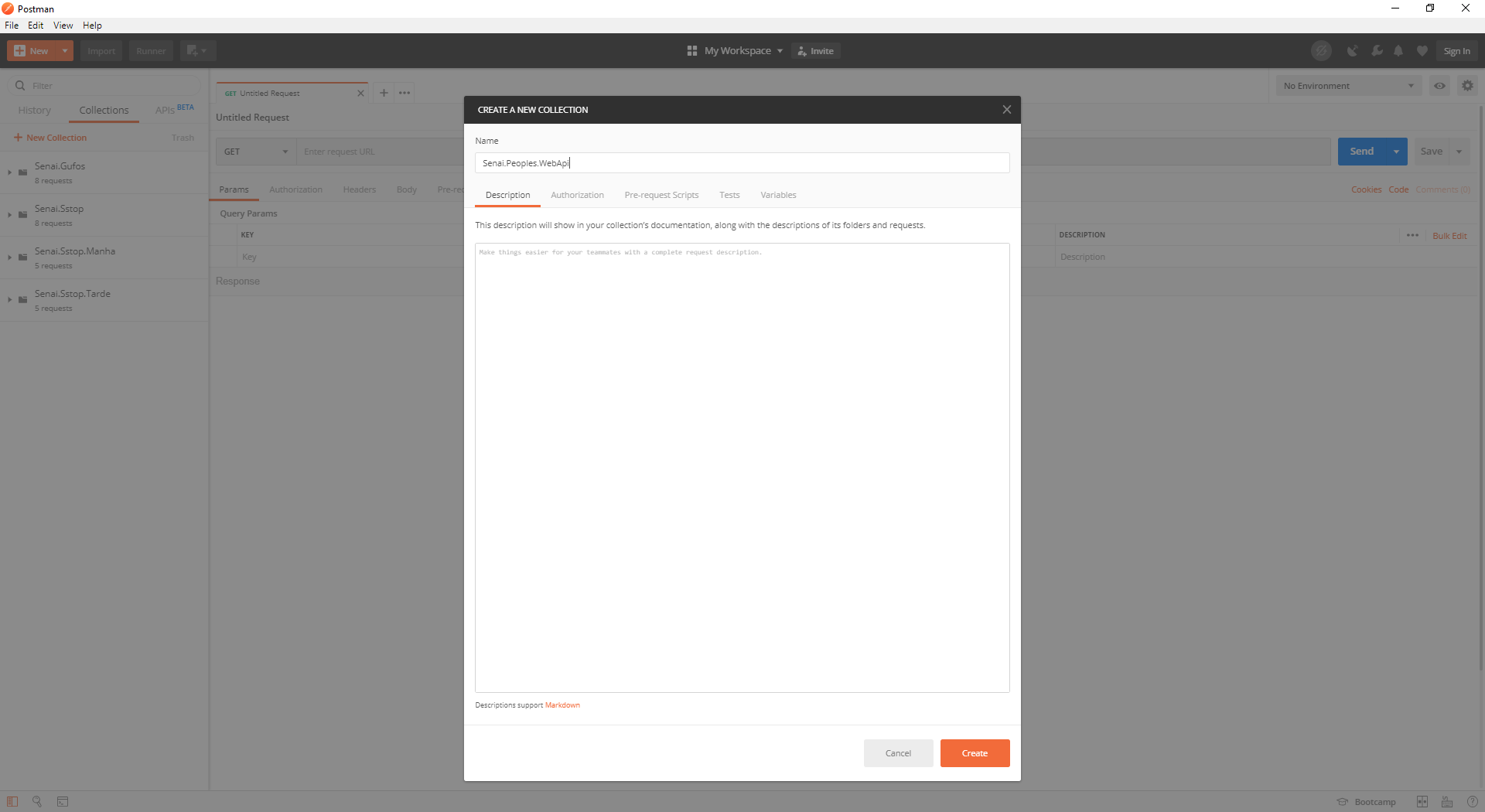
Uma vez que o controller foi criado, devemos criar um novo endpoint que seja capaz de receber essa requisição.



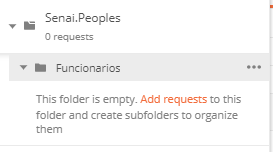
Além disso, como no repositório, o método de acesso ao banco já foi criado, criamos uma nova instância do Repositório a fim de ter acesso à função de listar todos os registros.

Como podemos testar a nossa estrutura?

Vamos até o postman, criamos uma coleção chamada Senai.Peoples.WebApi e adicionamos as requisições dentro da pasta correspondente.

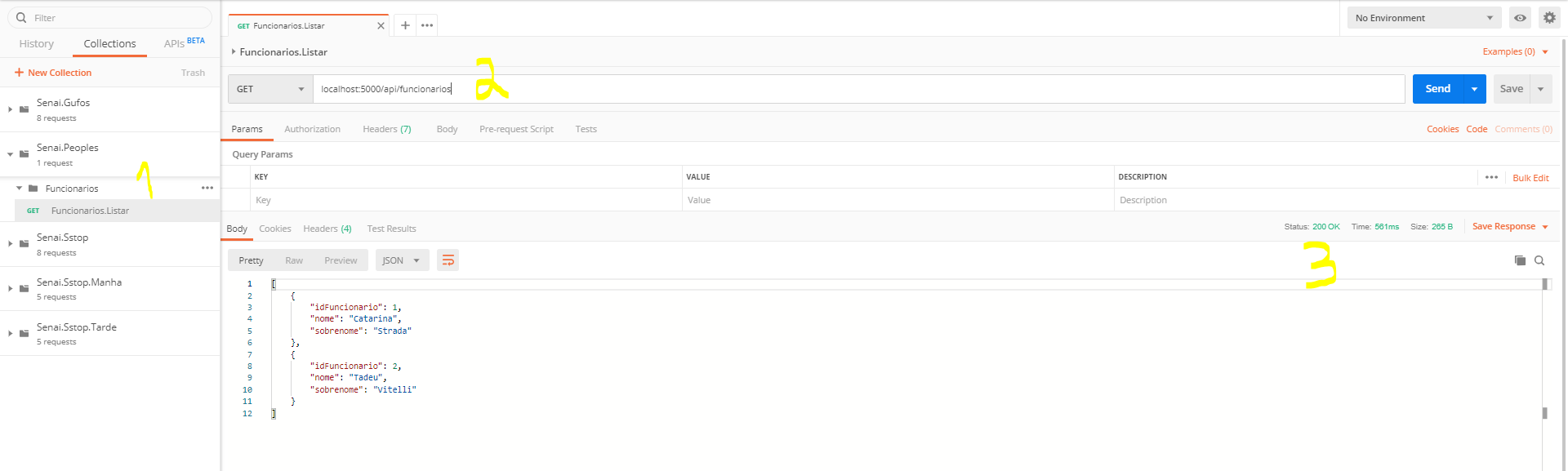


Criar uma pasta para os itens correspondentes.



Adicionar a requisições.

Quando estamos tratando de listar os dados desejados, e assim como incluímos no backend, trabalhamos com o verbo HTTP.



Ao salvarmos a requisição e executarmos, temos GET /api/funcionarios.

Ela irá nos retornar o código 200 uma vez que sua requisição foi feita com sucesso e os dados serão apresentados do banco de dados.

Agora, vamos partir para o próximo passo.

Vamos fazer a mesma solução para os outros itens propostos.

Repositório

BuscarPorId();

Controller

GET /api/funcionarios/{id}

Repositório

Cadastrar(FuncionarioDomain funcionario);

Controller

POST /api/funcionarios

Repositório

Atualizar(FuncionarioDomain funcionario);

Controller

PUT /api/funcionarios/{id}

Repositório

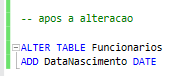
Deletar(int id);

Controller

DELETE /api/funcionarios/{id}

Vamos alterar o que foi proposto no problema.

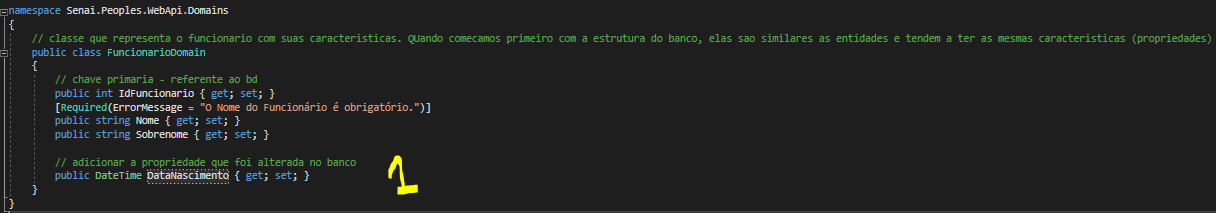
Incluir uma data de nascimento para o funcionário. Vamos alterar primeiro no banco de dados.



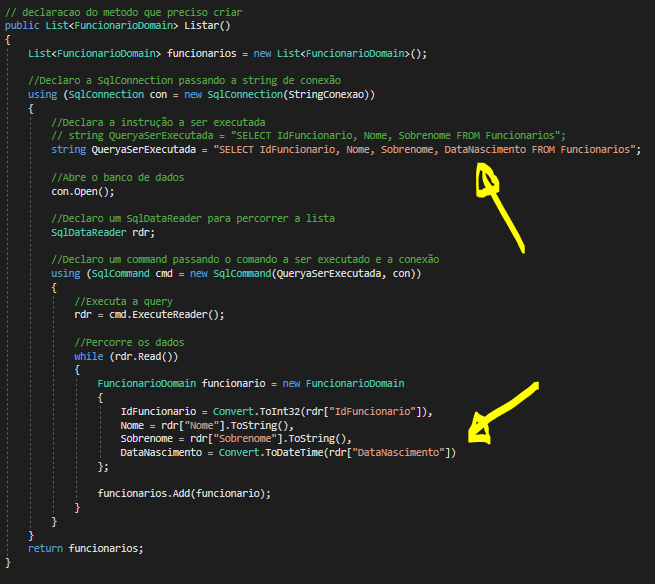
Atualizar todos os registros da tabela.



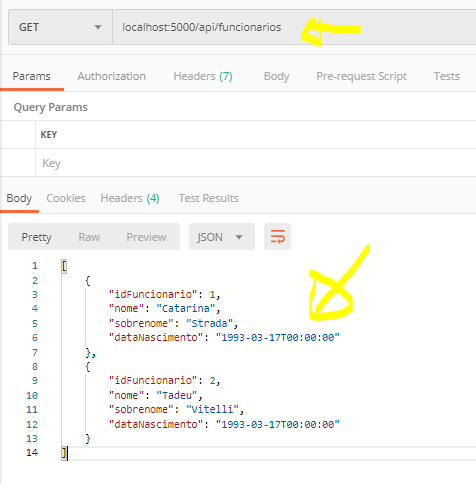
Aplicação do BackEnd.



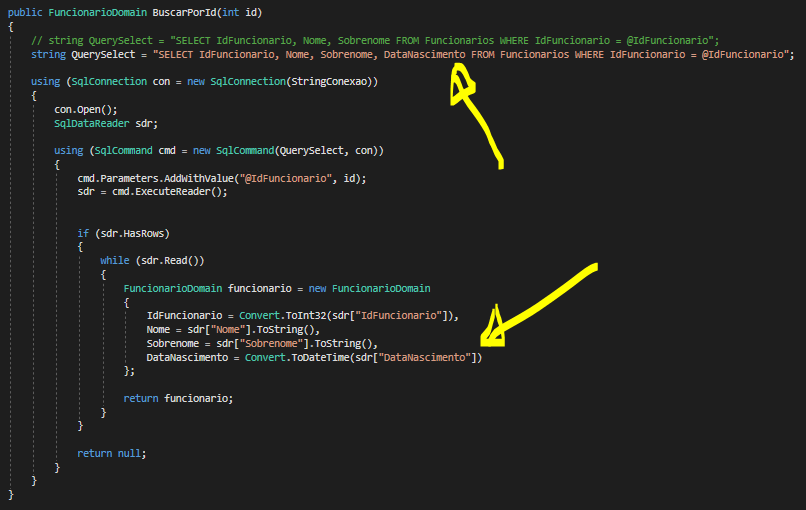
Uma vez que o modelo foi alterado, alterar a sua consulta na lista (repositório) para que essa informação seja apresentada na consulta.



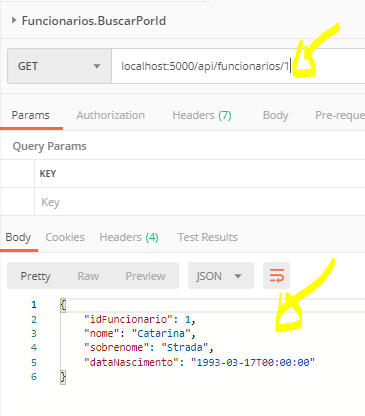
Executar o projeto e utilizar no postman a mesma consulta que fora feita anteriormente.



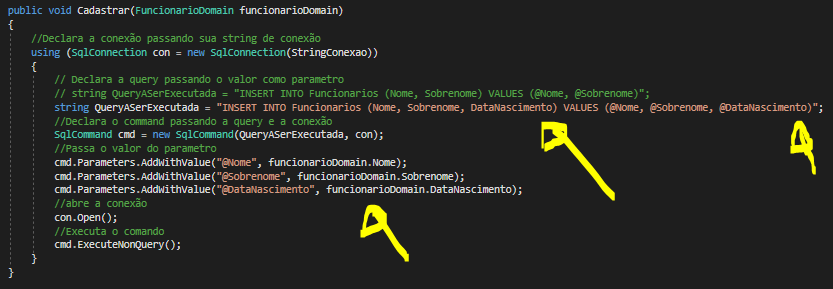
BuscarPorId()



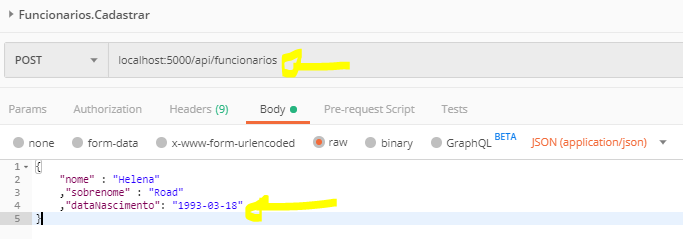
Postman – api/funcionarios/{id}



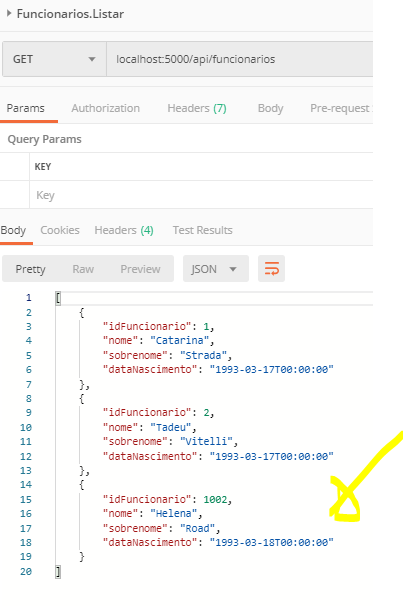
Cadastrar



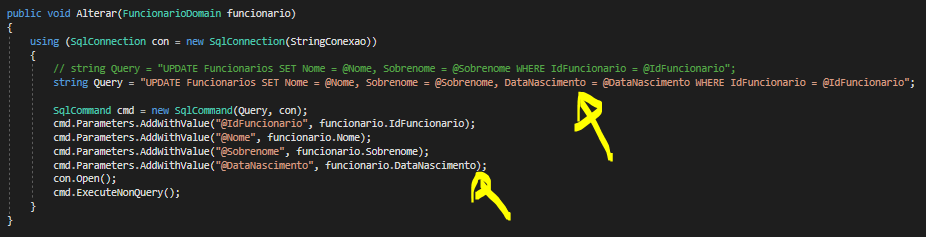
Postman



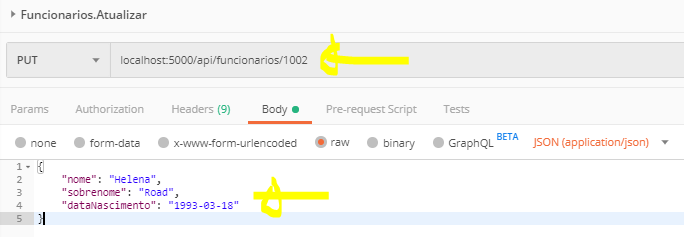
Vamos testar a lista para verificar se os valores foram preenchidos corretamente.



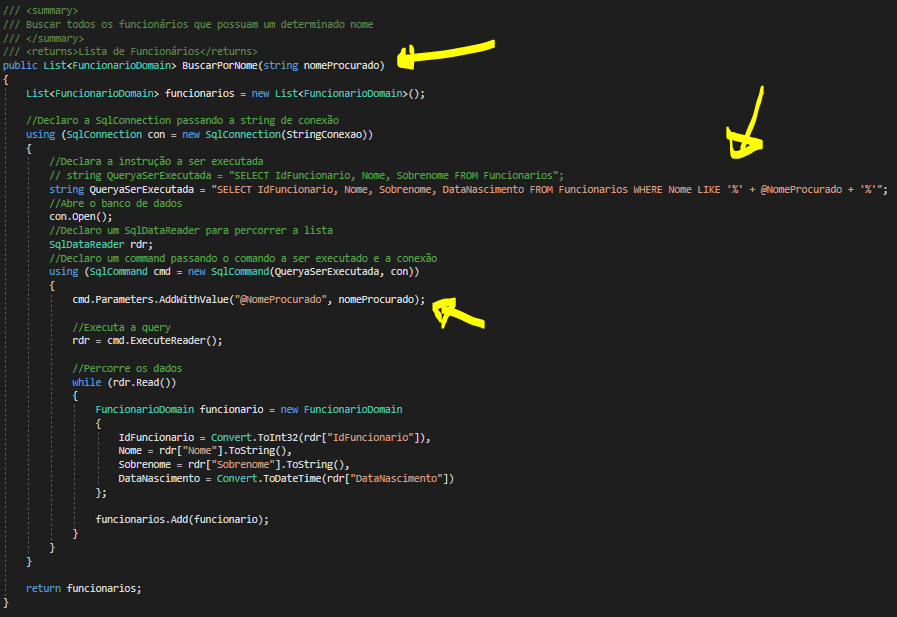
BackEnd – Atualizar



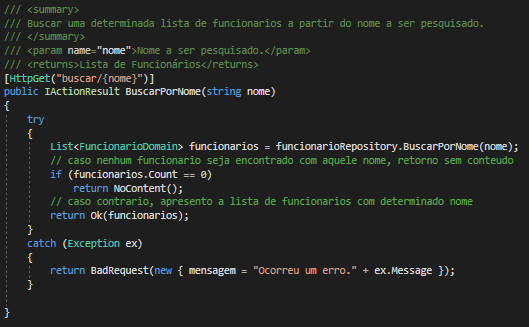
Postman



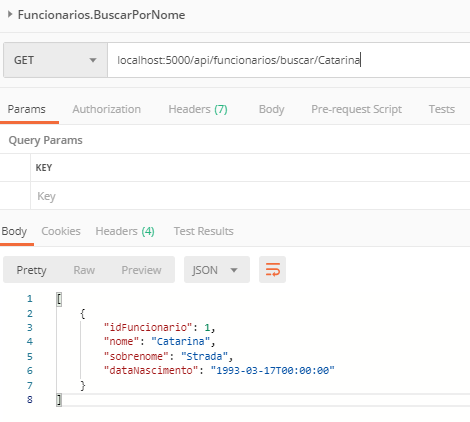
Vamos criar um novo endpoint para mostrar todos os funcionários que possuam um determinado nome.



Vamos alterar o controller.



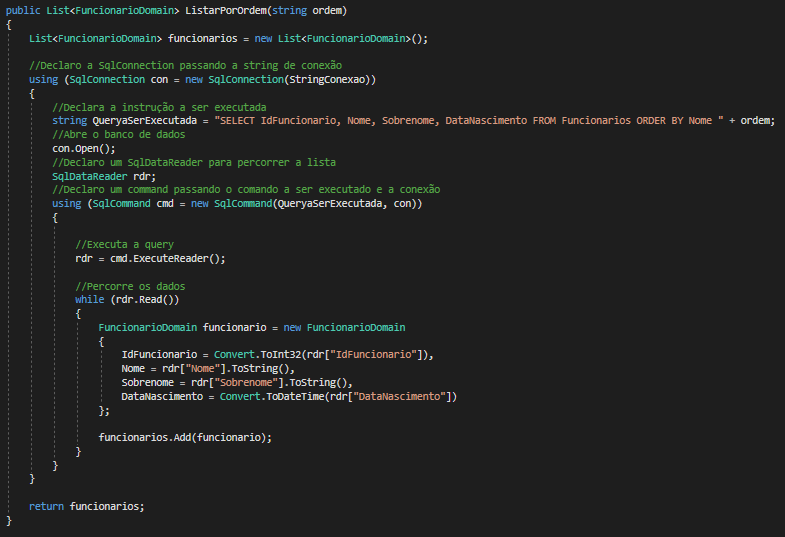
Postman



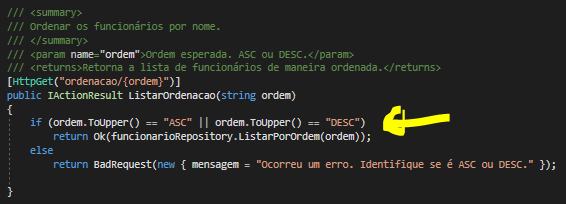
Pesquisando um nome que existe, ele irá me retornar a lista que segue. Caso eu não encontre registros, posso retornar o código de erro 204 a fim de informar para o usuário que nenhum conteúdo foi encontrado.



O último desafio era para apresentar uma lista ordenada dada uma entrada válida do usuário. Seja ela ascendente (asc) ou descendente (desc).



O interessante deste ponto, é ordenar a lista conforme o esperado e verificar que caso o usuário digite alguma informação que não seja válida, mostre uma mensagem de erro para o usuário.



Postman

