

# Atividade

## Criar o restante do CRUD de projetos, testando-o no Insomnia

Nesta atividade, você continuará a implementação dos códigos no projeto Web API da empresa **ExoApi**. Nessa etapa, sua tarefa será o desenvolvimento dos códigos nas classes **ProjetosController.cs** e **ProjetoRepository.cs** para os métodos **Cadastrar()**, **BuscarPorId()**, **Atualizar()** e **Deletar()**. O resultado dessa implementação será a finalização do CRUD (**Create**, **Read**, **Update** e **Delete**) de Projetos e um método de busca pelo ID.

Os testes nessa atividade serão diferentes da atividade anterior, pois serão realizados no aplicativo **Insomnia** ou **Postman**.

### Importante

Para realizar esse tutorial, é necessário baixar e descompactar o arquivo **ATIVIDADE-04.zip** disponível no AVA.



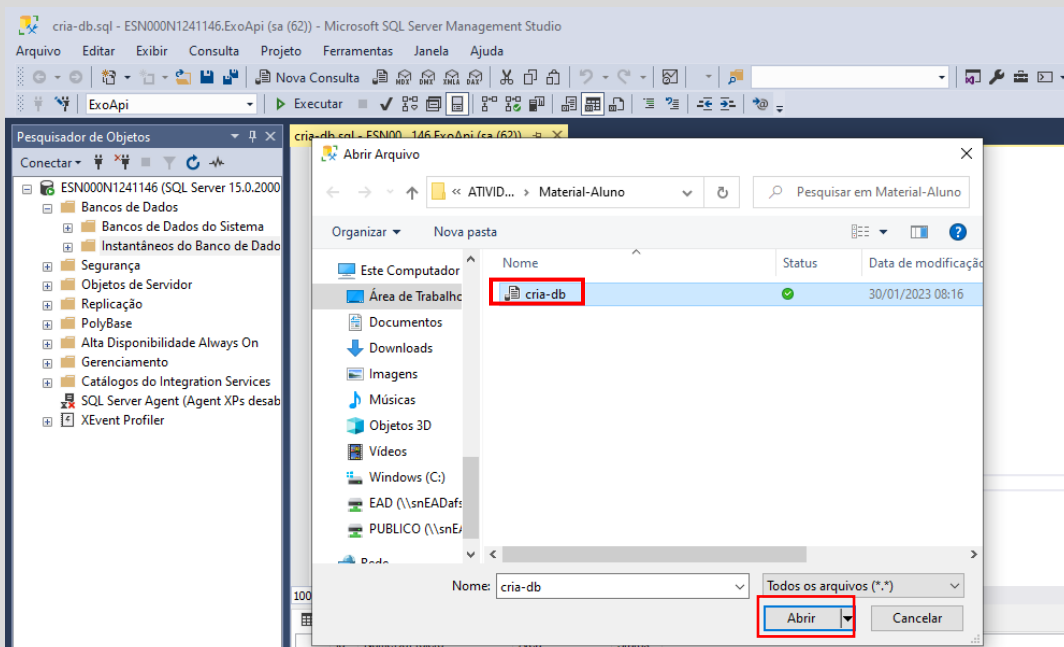
## Criando o banco no SSMS

### Importante

Os passos a seguir devem ser realizados caso você tenha apagado o banco da atividade anterior ou queira criar um novo banco. Caso já esteja com o banco criado, pule essa etapa e vá para [Preparação dos arquivos no VSCode](#).



1. Abra o SQL Server Management Studio (SSMS). Clique em **Arquivo > Abrir Arquivo...** e localize a pasta baixada para realizar a atividade. Na pasta **Material-Aluno**, selecione o script **cria-db.sql** e clique em **Abrir**.

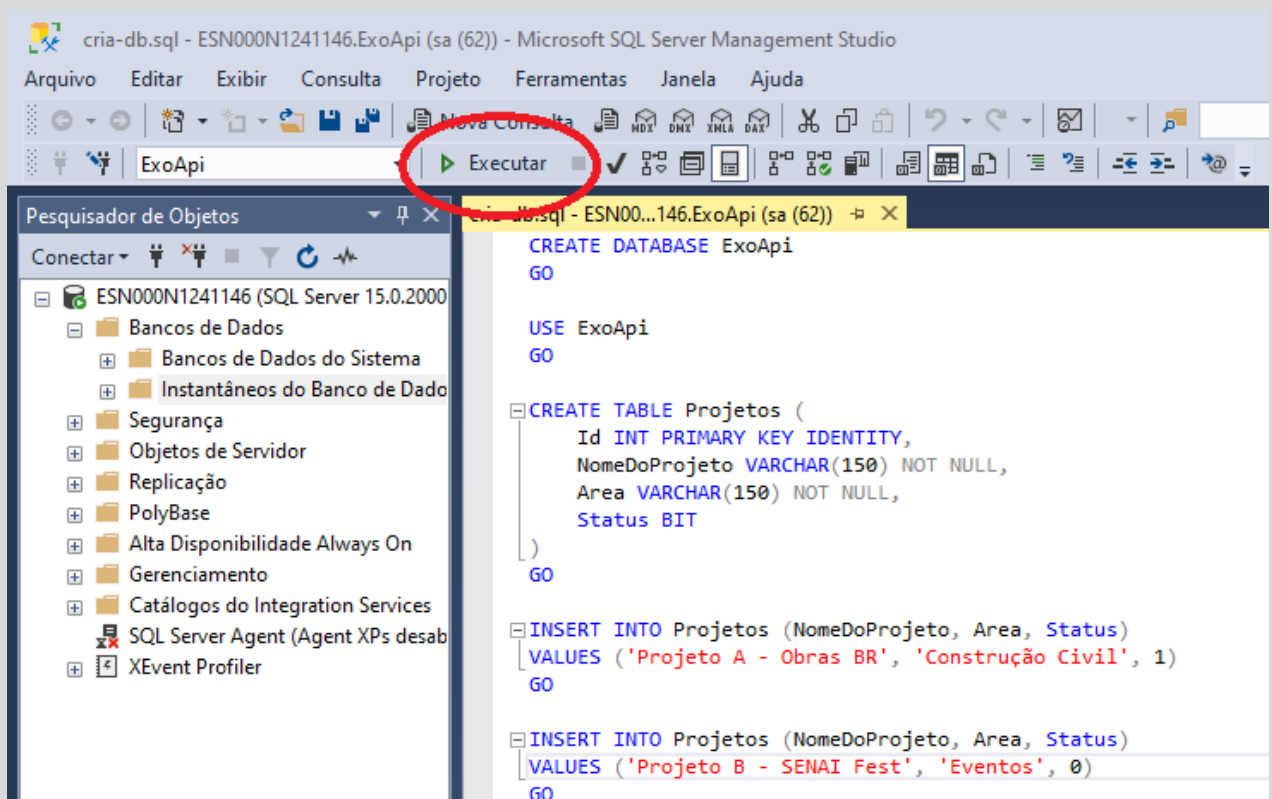


## Dica!

Caso já exista um banco de dados e você queira criar um novo, será necessário deletar o existente antes de executar esse script.



2. Com o arquivo **cria-db.sql** aberto, clique em **Executar** para executar o banco e criar os usuários.



## Preparação dos arquivos no VSCode

### Importante

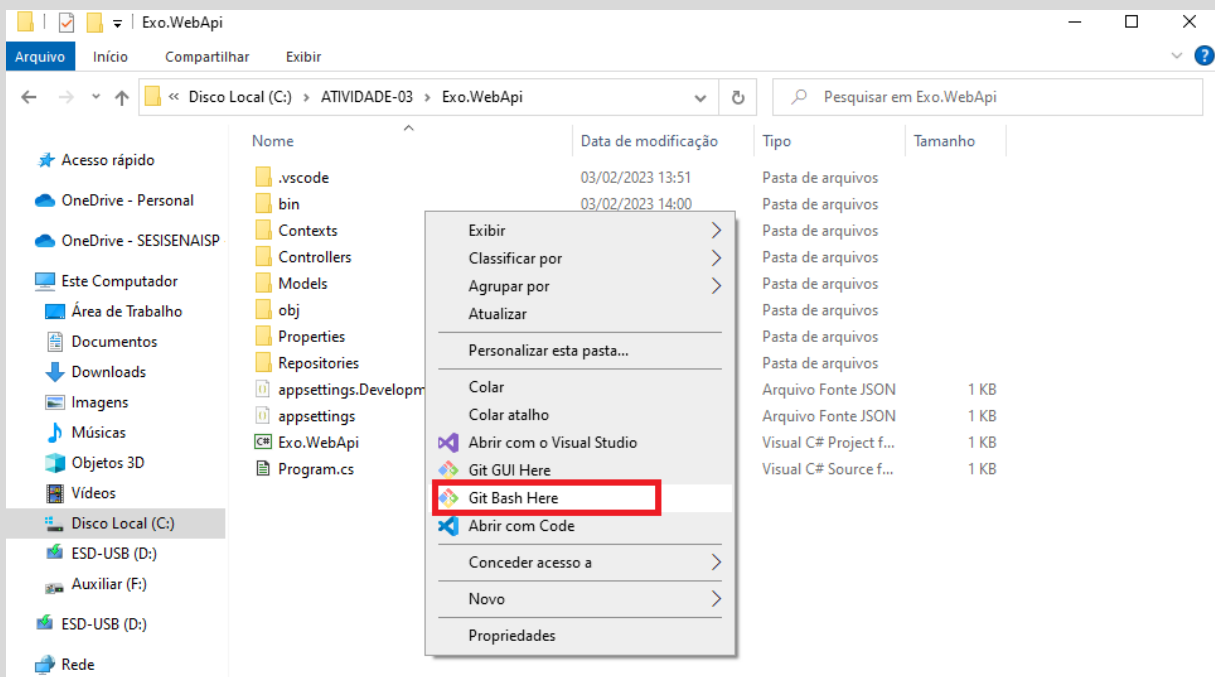
O projeto de API dessa atividade será o mesmo da antecedente. Na etapa anterior, foram realizadas algumas implementações e continuaremos com esse mesmo projeto.

Caso seja necessário, os arquivos estão anexados à atividade (na parte em que paramos).

Caso você queira usar o seu projeto, pule a próxima etapa e vá diretamente para [Complementação dos códigos das classes](#).



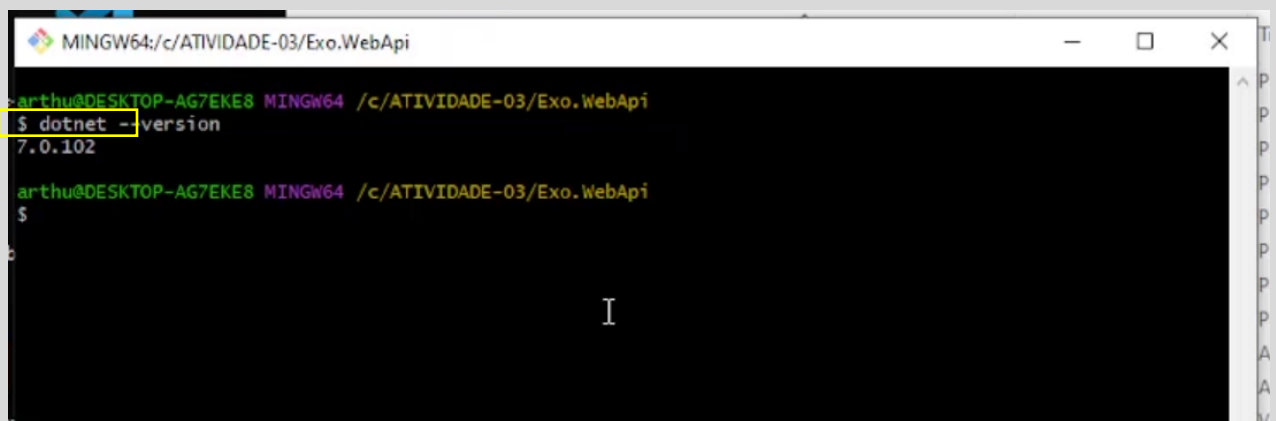
1. Abra a pasta que você usou para baixar a atividade e localize a **Exo.WebApi**. Clique com o botão direito dentro dela e escolha **Git Bash Here** para abrir o terminal.



2. No terminal que será aberto, digite o comando abaixo e dê **Enter** para verificar a versão do dotnet instalada em sua máquina.

```
dotnet --version
```

3. Certifique-se que sua versão do dotnet seja a 6 ou superior. No nosso caso, a versão é 7.0.102, como mostra a figura.

A screenshot of a terminal window titled 'MINGW64:/c/ATIVIDADE-03/Exo.WebApi'. The prompt is 'arthu@DESKTOP-AG7EKE8 MINGW64 /c/ATIVIDADE-03/Exo.WebApi'. The command '\$ dotnet --version' is entered, and the output '7.0.102' is displayed. The prompt '\$' is shown again on the next line.

```
arthu@DESKTOP-AG7EKE8 MINGW64 /c/ATIVIDADE-03/Exo.WebApi
$ dotnet --version
7.0.102
arthu@DESKTOP-AG7EKE8 MINGW64 /c/ATIVIDADE-03/Exo.WebApi
$
```

4. Agora, digite o comando abaixo e dê **Enter** para abrir o VSCode com o projeto já aberto.

```
code .
```

## Complementação dos códigos das classes

1. Abra o arquivo **ProjetoRepository.cs** e substitua o código pelo seguinte:

```
using Exo.WebApi.Contexts;  
using Exo.WebApi.Models;  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Threading.Tasks;  
  
namespace Exo.WebApi.Repositories  
{  
    public class ProjetoRepository  
    {  
        private readonly ExoContext _context;  
        public ProjetoRepository(ExoContext context)  
        {  
            _context = context;  
        }  
        public List<Projeto> Listar()  
        {  
            return _context.Projetos.ToList();  
        }  
  
        // Código novo que completa o CRUD.  
        public void Cadastrar(Projeto projeto)  
        {  
            _context.Projetos.Add(projeto);  
            _context.SaveChanges();  
        }  
        public Projeto BuscarporId(int id)  
        {  
            return _context.Projetos.Find(id);  
        }  
    }  
}
```

Continuação do código na próxima página >>>

>>> continuação do código da página anterior

```
public void Atualizar(int id, Projeto projeto)
{
    Projeto projetoBuscado = _context.Projetos.Find(id);
    if (projetoBuscado != null)
    {
        projetoBuscado.NomeDoProjeto = projeto.NomeDoProjeto;
        projetoBuscado.Area = projeto.Area;
        projetoBuscado.Status = projeto.Status;
    }
    _context.Projetos.Update(projetoBuscado);
    _context.SaveChanges();
}
public void Deletar(int id)
{
    Projeto projetoBuscado = _context.Projetos.Find(id);
    _context.Projetos.Remove(projetoBuscado);
    _context.SaveChanges();
}
}
```

## Importante

A atividade solicita que você implemente mais quatro métodos em seu projeto. Devido ao uso do Repository padrão, cada método implementado na classe **ProjetosController.cs** deverá ter outro de mesmo nome na classe **ProjetoRepository.cs**. Por exemplo, haverá dois métodos **Deletar()**, sendo um em **ProjetosController.cs** e outro em **ProjetoRepository.cs**, porém, suas sintaxes serão diferentes.



2. Abra o arquivo **ProjetosController.cs** e substitua o código pelo que está a seguir:

```
using Exo.WebApi.Models;
using Exo.WebApi.Repositories;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System;

namespace Exo.WebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProjetosController : ControllerBase
    {
        private readonly ProjetoRepository _projetoRepository;
        public ProjetosController(ProjetoRepository
projetoRepository)
        {
            _projetoRepository = projetoRepository;
        }

        [HttpGet]
        public IActionResult Listar()
        {
            return Ok(_projetoRepository.Listar());
        }

        // Código novo que completa o CRUD.
        [HttpPost]
        public IActionResult Cadastrar(Projeto projeto)
        {
            _projetoRepository.Cadastrar(projeto);
            return StatusCode(201);
        }
    }
}
```

Continuação do código na próxima página >>>



>>> continuação do código da página anterior

```
[HttpGet("{id}")]
public IActionResult BuscarPorId(int id)
{
    Projeto projeto = _projetoRepository.BuscarporId(id);
    if (projeto == null)
    {
        return NotFound();
    }
    return Ok(projeto);
}

[HttpPut("{id}")]
public IActionResult Atualizar(int id, Projeto projeto)
{
    _projetoRepository.Atualizar(id, projeto);
    return StatusCode(204);
}

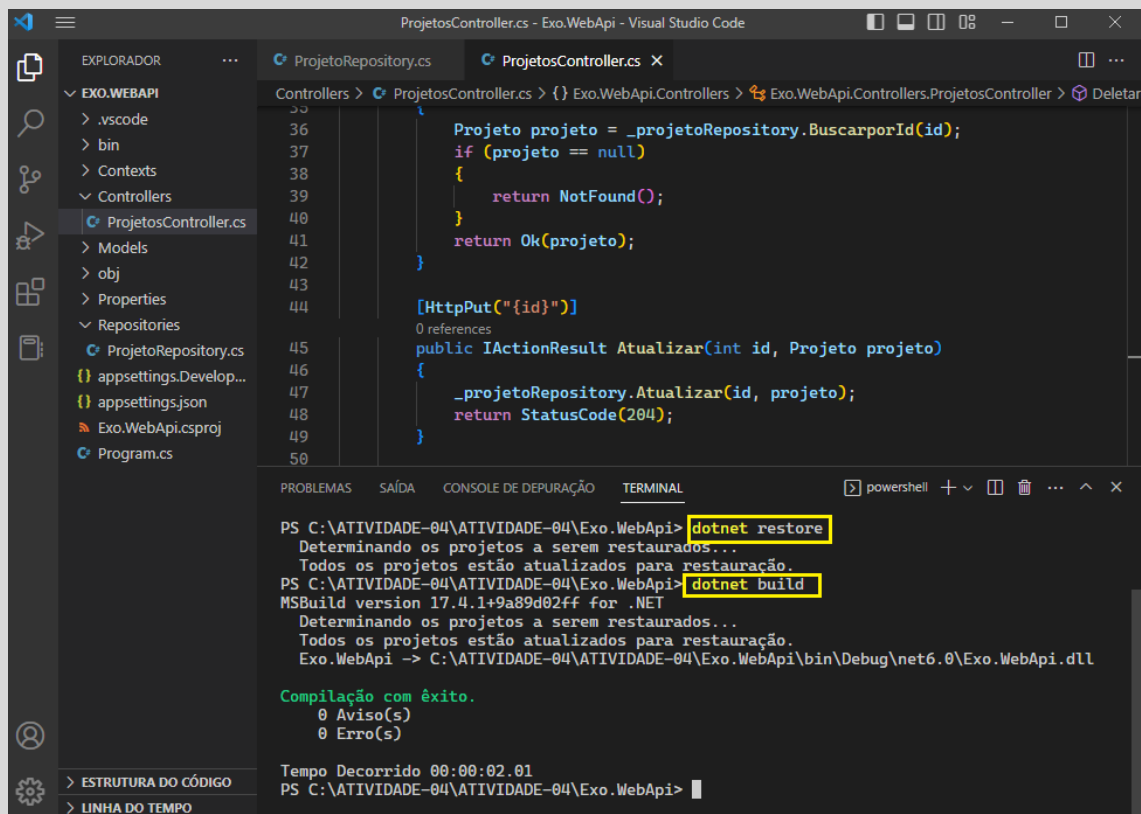
[HttpDelete("{id}")]
public IActionResult Deletar(int id)
{
    try
    {
        _projetoRepository.Deletar(id);
        return StatusCode(204);
    }
    catch (Exception e)
    {
        return BadRequest();
    }
}
}
```

3. Nesse projeto, já existem alguns pacotes instalados. Abra o terminal (menu **Terminal** > **Novo Terminal**) e, nele, digite o comando abaixo para restaurar o projeto e consolidar a instalação:

```
dotnet restore
```

4. Digite o comando abaixo para compilar a aplicação.

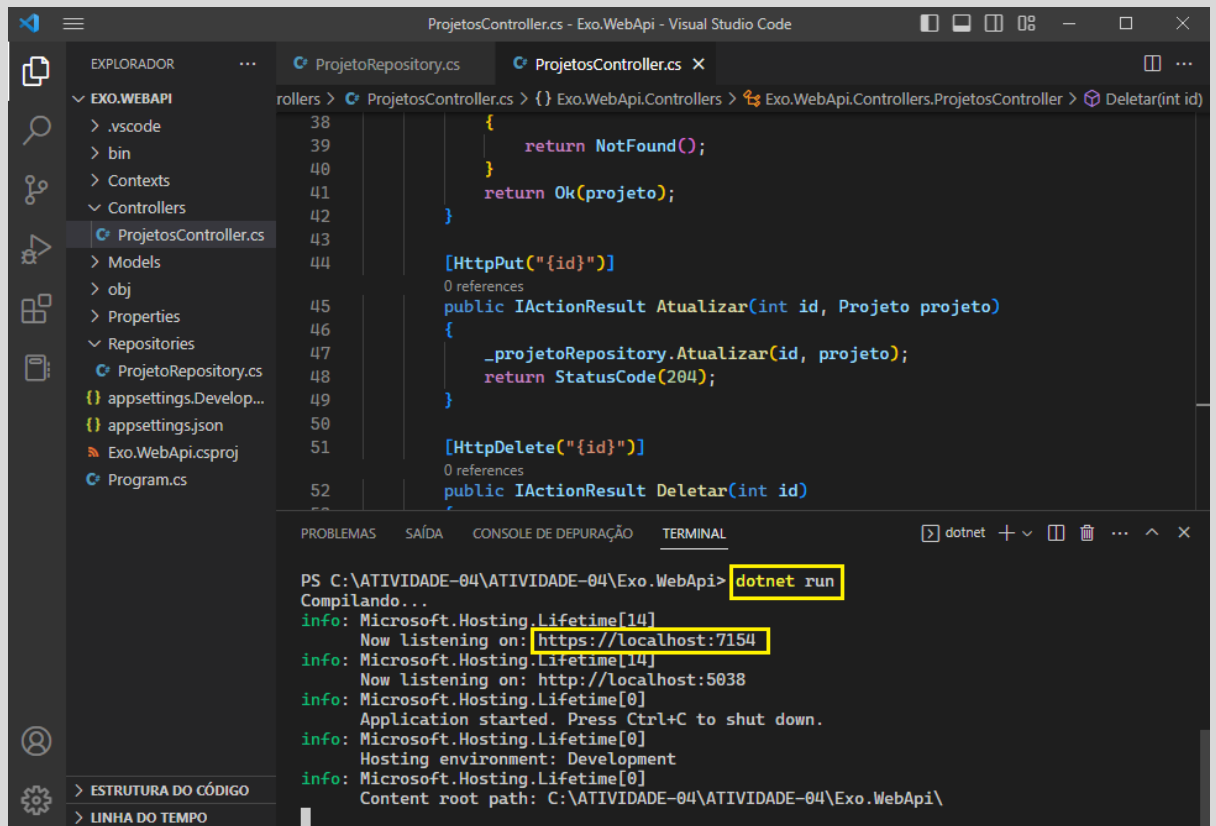
```
dotnet build
```



5. Digite o próximo comando para executar o projeto.

```
dotnet run
```

6. Anote o endereço que apareceu no terminal. Utilizaremos esse link seguido do sufixo **api/projetos** para realizar a próxima etapa da atividade.



The screenshot shows the Visual Studio Code interface with the following components:

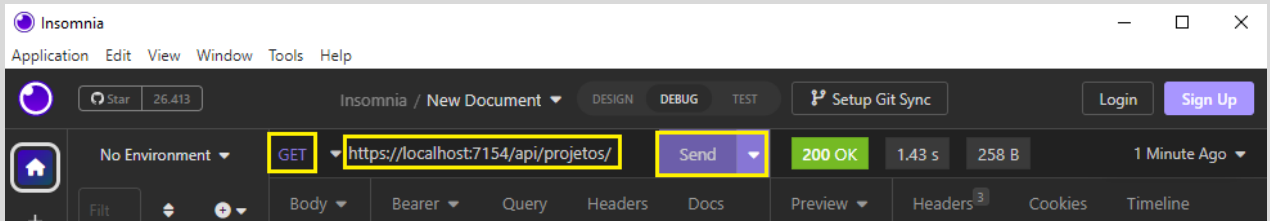
- EXPLORADOR (Explorer):** Displays the project structure for EXO.WEBAPI, including folders like .vscode, bin, Contexts, Controllers, Models, obj, Properties, Repositories, and files like appsettings.Develop..., appsettings.json, Exo.WebApi.csproj, and Program.cs.
- PROJETOSCONTROLLER.CS:** The active file in the editor, showing the following code:

```
38 {  
39     return NotFound();  
40 }  
41 return Ok(projeto);  
42 }  
43  
44 [HttpPut("{id}")]  
45 0 references  
46 public IActionResult Atualizar(int id, Projeto projeto)  
47 {  
48     _projetoRepository.Atualizar(id, projeto);  
49     return StatusCode(204);  
50 }  
51  
52 [HttpDelete("{id}")]  
53 0 references  
54 public IActionResult Deletar(int id)
```
- TERMINAL:** Shows the output of the `dotnet run` command, indicating the application is running and listening on `https://localhost:7154` and `http://localhost:5038`. The output includes:

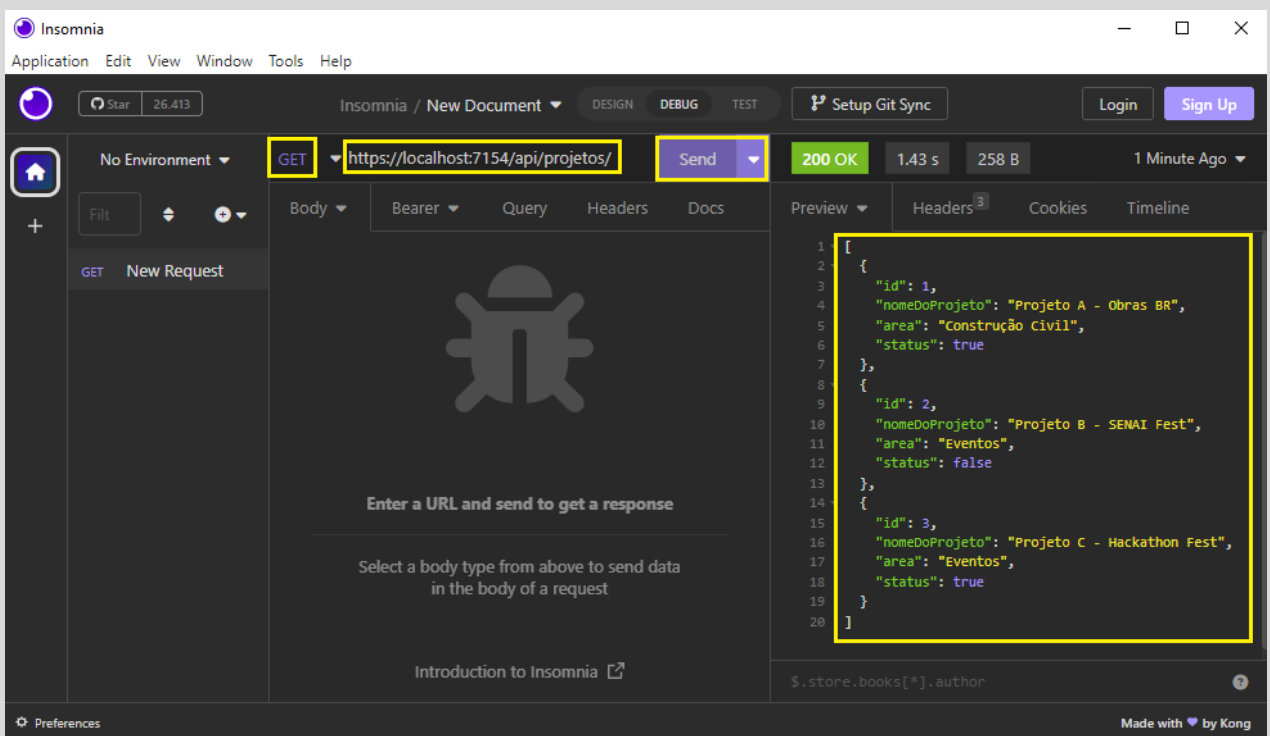
```
PS C:\ATIVIDADE-04\ATIVIDADE-04\Exo.WebApi> dotnet run  
Compilando...  
info: Microsoft.Hosting.Lifetime[14]  
      Now listening on: https://localhost:7154  
info: Microsoft.Hosting.Lifetime[14]  
      Now listening on: http://localhost:5038  
info: Microsoft.Hosting.Lifetime[0]  
      Application started. Press Ctrl+C to shut down.  
info: Microsoft.Hosting.Lifetime[0]  
      Hosting environment: Development  
info: Microsoft.Hosting.Lifetime[0]  
      Content root path: C:\ATIVIDADE-04\ATIVIDADE-04\Exo.WebApi\
```

## Utilizando o Insomnia

1. Baixe o **Insomnia** em <https://insomnia.rest/download>. Abra o programa e configure os campos conforme indicado abaixo.



2. O resultado aparecerá na janela à direita.



### Dica!

O **200 OK** que aparece ao lado do botão **Send** é o código de resposta da requisição bem-sucedida. Caso ocorra algum problema, a resposta seria outro código.

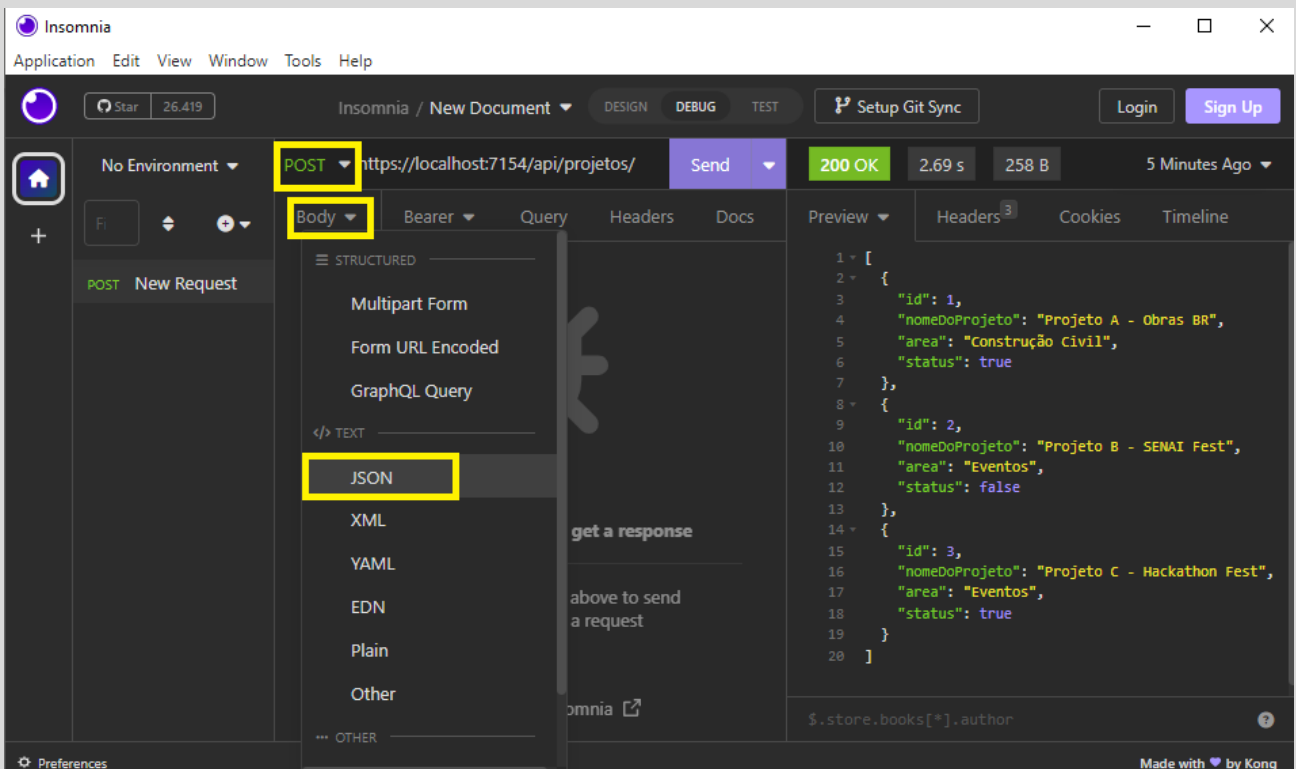


## Você sabia?

Note que o resultado foi o mesmo da atividade anterior. Isso ocorre, porque esses dados fornecidos pela API podem ser visualizados – tanto quando chamamos via barra de endereços do **navegador**, quanto por alguma **página formatada** com front-end ou nos softwares que testam API como o **Insomnia** e o **Postman**.

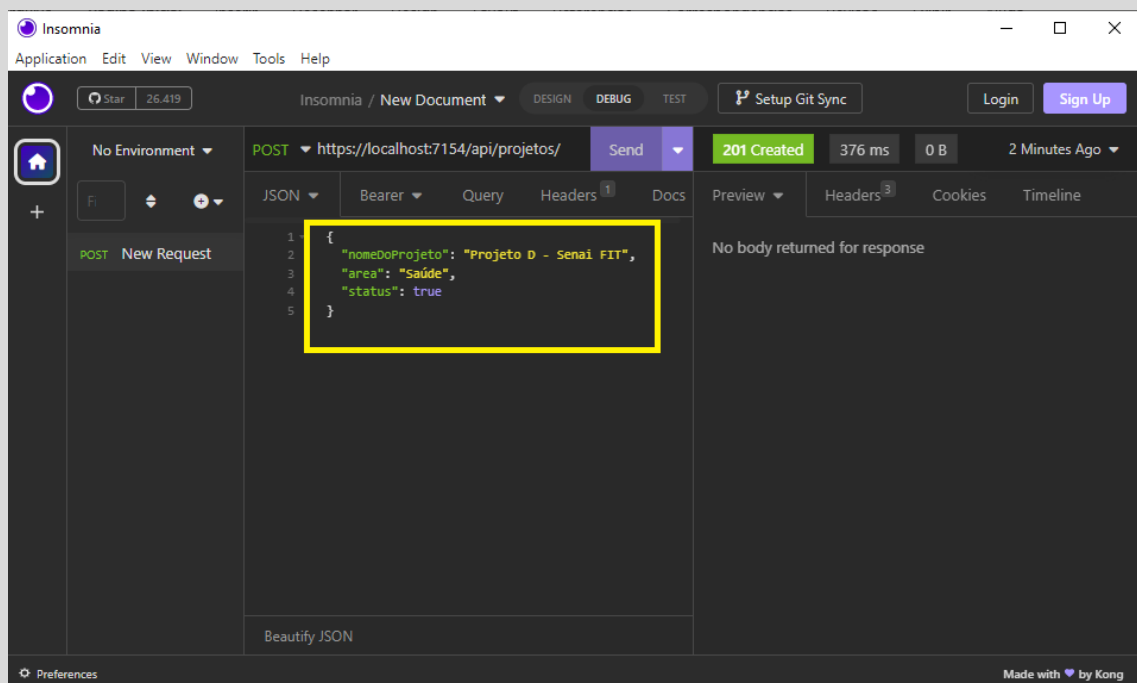


3. Agora, teste o método Cadastrar (POST). Escolha o método **POST**, insira a url **https://localhost:7154/api/projetos/** e, em Body, troque para JSON, conforme a imagem:



4. Na janela **JSON**, insira o código abaixo para inserção de um novo cadastro no banco de dados da aplicação e clique em **SEND**. Aparecerá o código **201 Created** em verde, como na imagem abaixo.

```
{  
  "nomeDoProjeto": "Projeto D - Senai FIT",  
  "area": "Saúde",  
  "status": true  
}
```

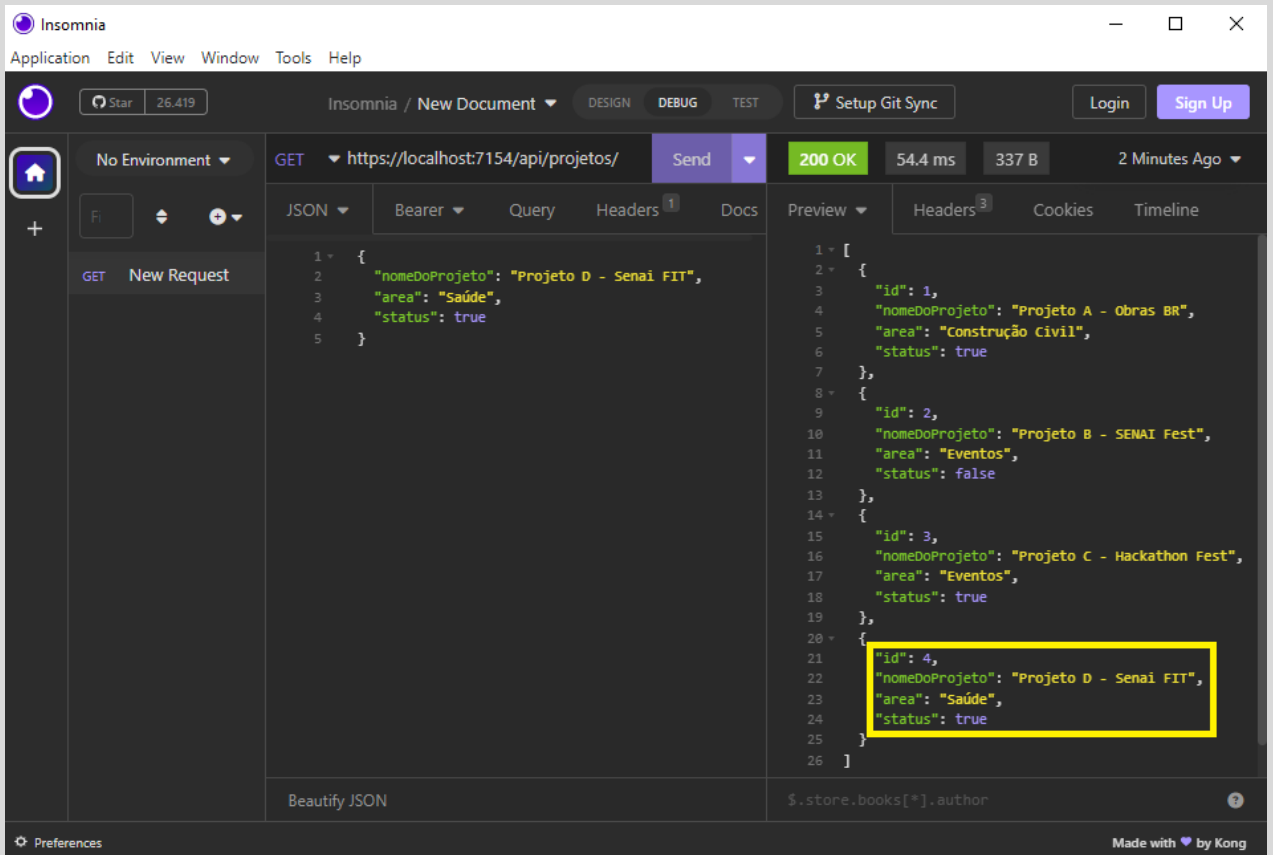


### Importante

O campo **ID** não é inserido. Ele está configurado para autoincremento e, por isso, será colocado automaticamente.



5. Você pode consultar a listagem dos itens do banco de dados com o novo projeto alterando o método para GET e clicando em **SEND**. Eles aparecerão na janela lateral, conforme a imagem.



### Você sabia?

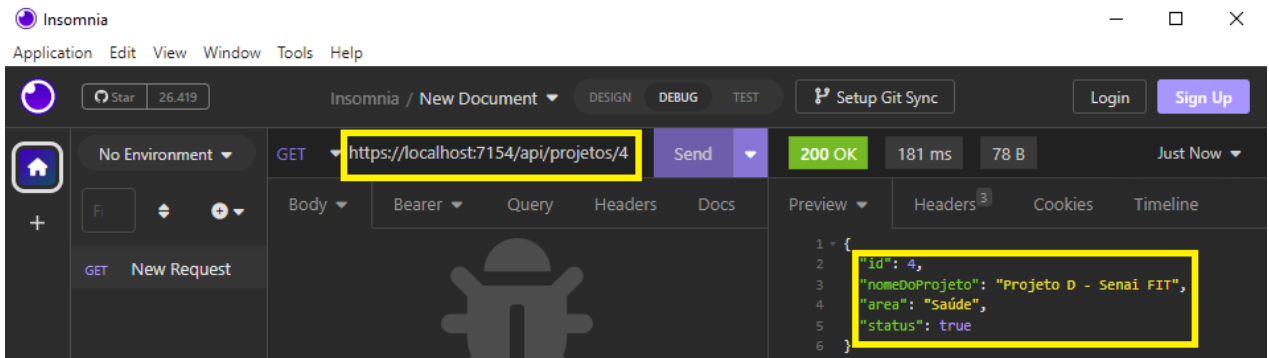
Você também pode consultar a listagem dos itens do banco de dados diretamente no **SSMS**.



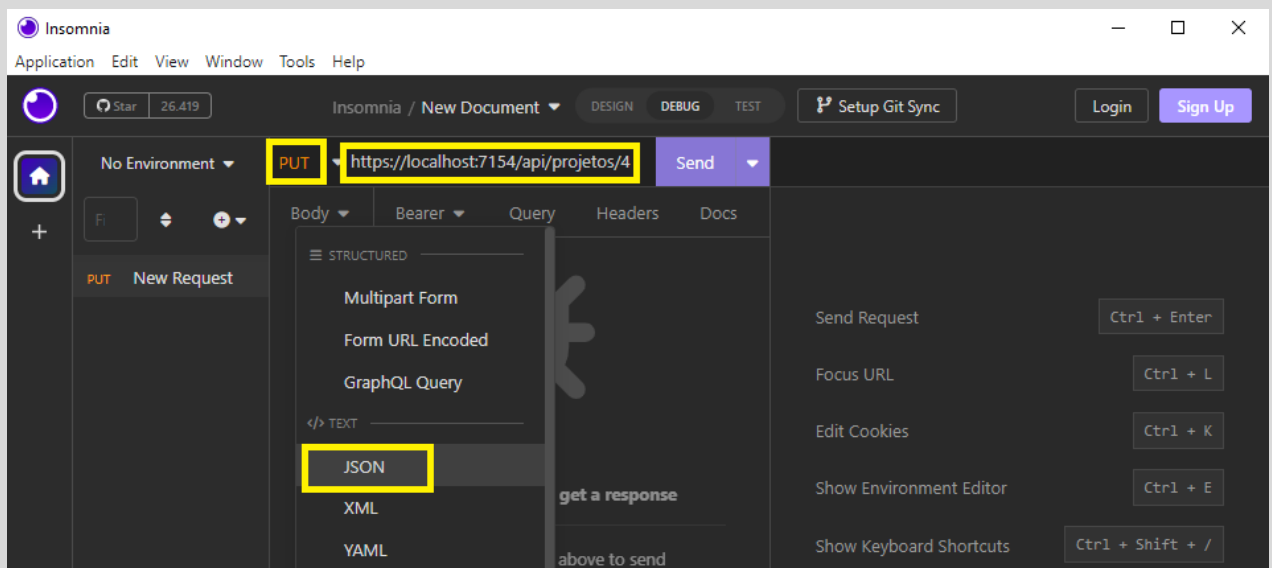
## Dica!

Também é possível usar o **GET** para visualizar as ids individualmente. Para isso, insira a url:

<https://localhost:7154/api/projetos/4> (na qual **4** pode ser outro registro que se deseja consultar) e clique em **SEND**. Nesse exemplo, o resultado será o quarto item do banco.



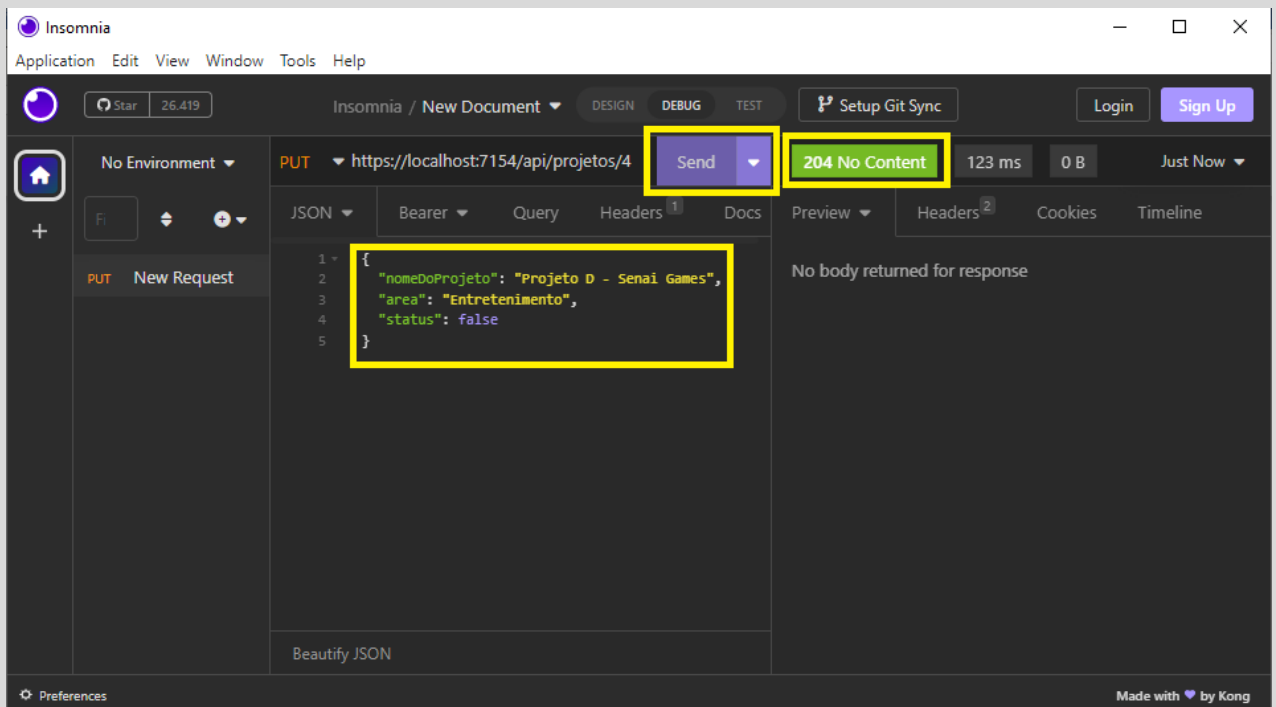
6. Agora, teste o método Atualizar (PUT) para alterar o último projeto que inserimos. Escolha o método PUT, insira a url: <https://localhost:7154/api/projetos/4> e mude Body para JSON, conforme a imagem.





7. Para atualização do item 4, substitua o código para o que está abaixo e clique em **SEND**. O resultado será o **código 204** (a solicitação foi bem-sucedida), conforme programamos na função **Atualizar** da classe **ProjetosController.cs**.

```
{  
    "nomeDoProjeto": "Projeto D - Senai Games",  
    "area": "Entretenimento",  
    "status": false  
}
```

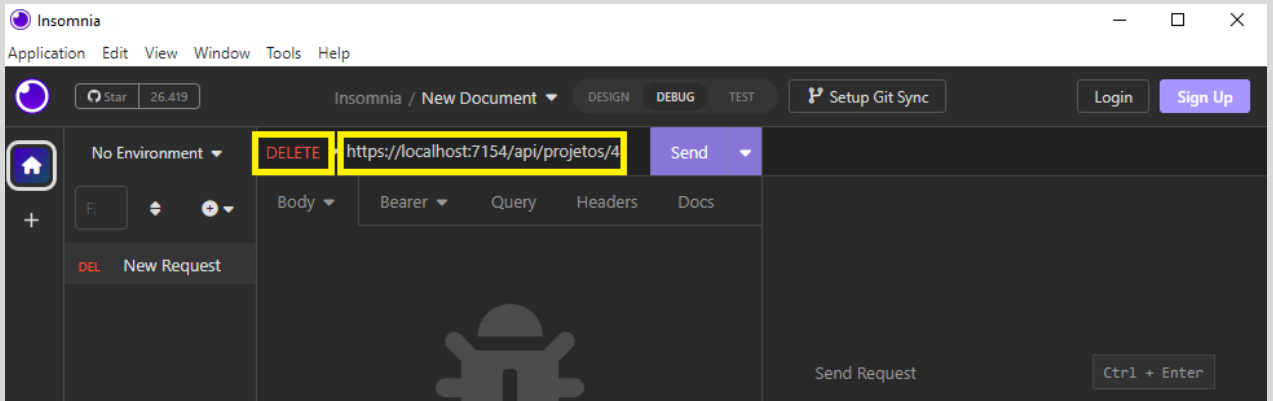


## Importante

No exemplo, a url tem o identificador 4 no final, mas você pode atualizar outro projeto do seu banco.



8. Por último, testaremos a remoção no Insomnia. Escolha o método DELETE, insira a url **https://localhost:7154/api/projetos/4** e clique em **SEND**. Será deletado o projeto com o id 4.



9. Você pode consultar a listagem dos itens do banco de dados sem o projeto deletado inserindo a url **https://localhost:7154/api/projetos/** e clicando em **SEND**. Eles aparecerão na janela lateral, exceto o projeto deletado, conforme a imagem.

