MODULO 5 - CQRS – Comand Query Responsibility Segregation

- Padrão de consulta que separa as Querys das ações que alteram o estado do sistema(Commands)

- Implementado de diversas maneiras, mas geralmente fica na camada de aplicação. Também existem implementações dele com um projeto para Commands e outro para Queries.

- Devido a essa separação logica (Comandos e consultas) facilita o uso de mais de um banco de dados. Porem essa decisão vem com diversos problemas como consistência de dados; (Usar SQL Server e MongoDB mas precisa de uma sincronia de dados)

- Melhora a legibilidade da aplicação, além de permitir maior separação das resposnabilidades e estimula a separação dos modelos;

COMMANDS: Representam ações no sistema, que realizam alterações em seu estado;(Cadastro projeto, inicialização..)

- Podem ser mapeadas de 1 para 1 a partir dos métodos de serviços de aplicação;

- Geralmente são nomeados com o sufixo Command, EX: CreatedProjectCommand;

- Commands contém as informações necessárias para a ação, sendo similar ao modelo de entrada utilizado pelo serviço de aplicação;

- Para cada Command deverá existir um CommandHandler, que é a Classe que lida com aquele comando; (Handler quem vai cuidar dos dados, gerenciar);

QUERIES: Representam consultas no sistema, que NÃO realizam alterações no estado dele;

- Podem ser mapeadas de 1 para 1 a partir dos métodos de serviços da aplicação;

- Geralmente são nomeadas com o sufixo Query. EX: GetAllSkillsQuery;

- As Queries contem informações necessárias para a consulta. EX: Contendo o identificador ou parâmetros de busca;

- Para cada Query deverá existir um QueryHandler(Processa e retorna a requisição), que é a Classe que lida com aquela consulta;

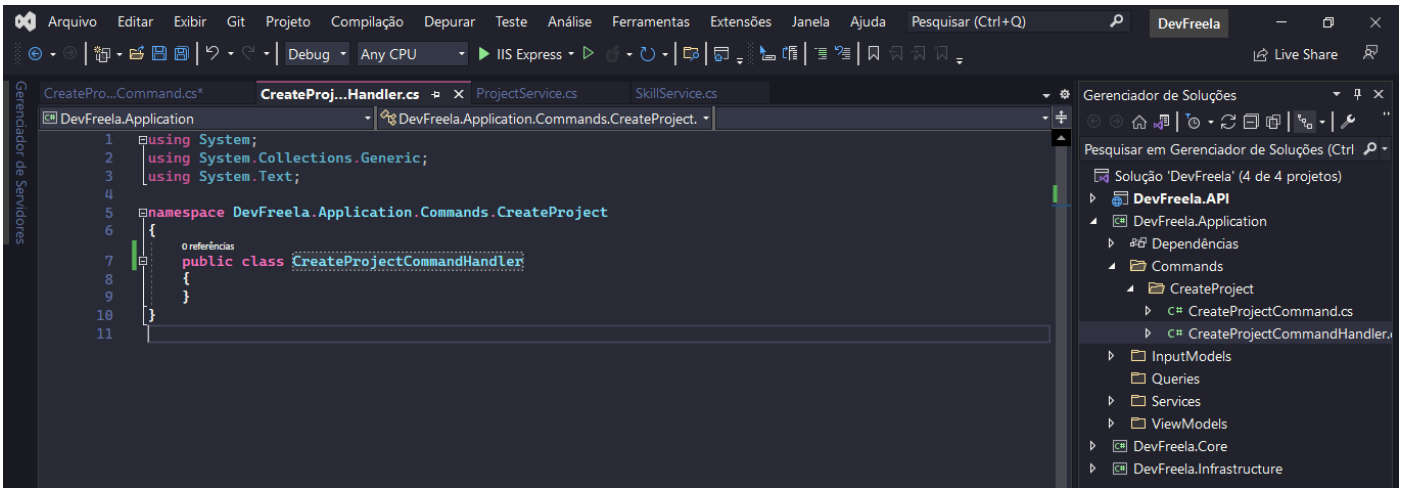
MEDIATR (Biblioteca Mediator o objeto central mediator gerencia)

- Implementa o padrão mediator, oferecendo uma espécie de mensageria interna em memoria;

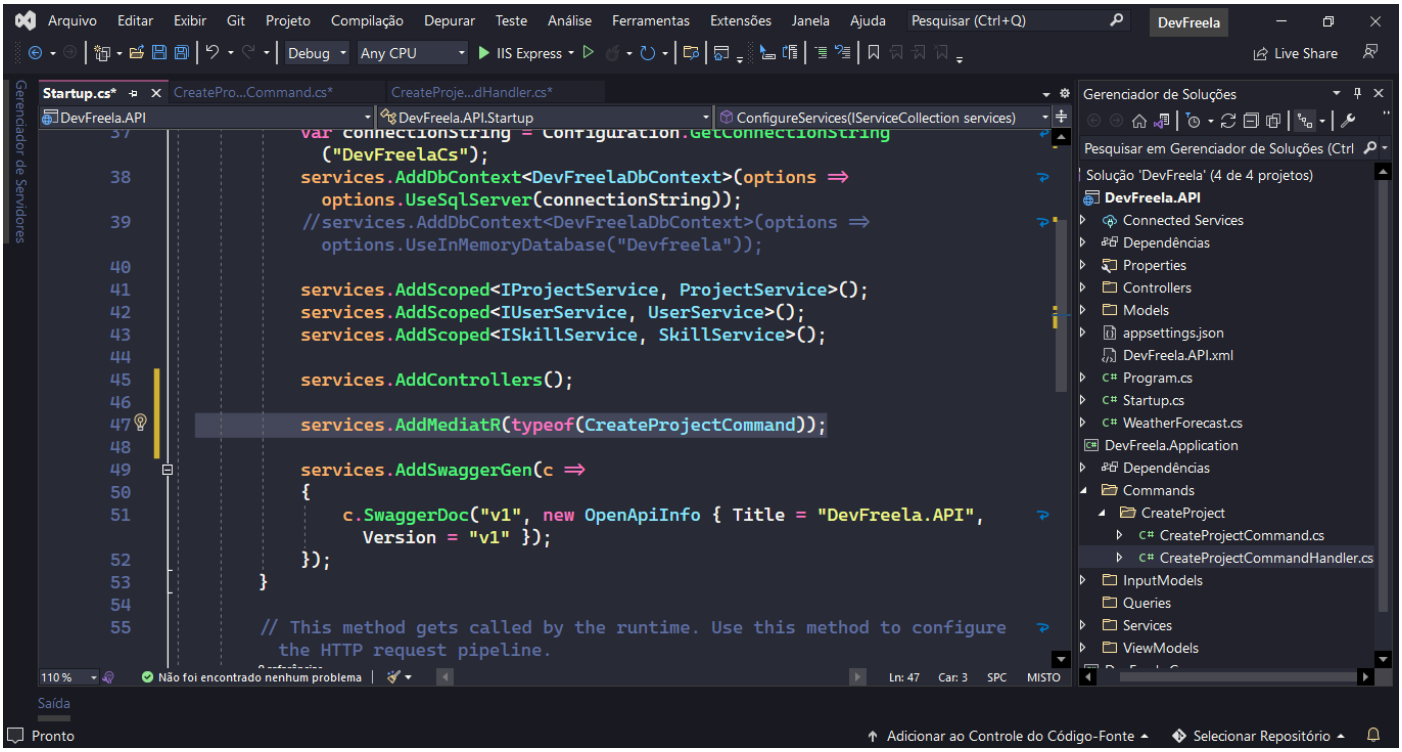
- Tem suporte a Commands e Queries, delegando a eles para serem processados pelos seus Handlers;

- Pacote MediatR e MediatR.Extensions.Microsoft.DependencyInjection

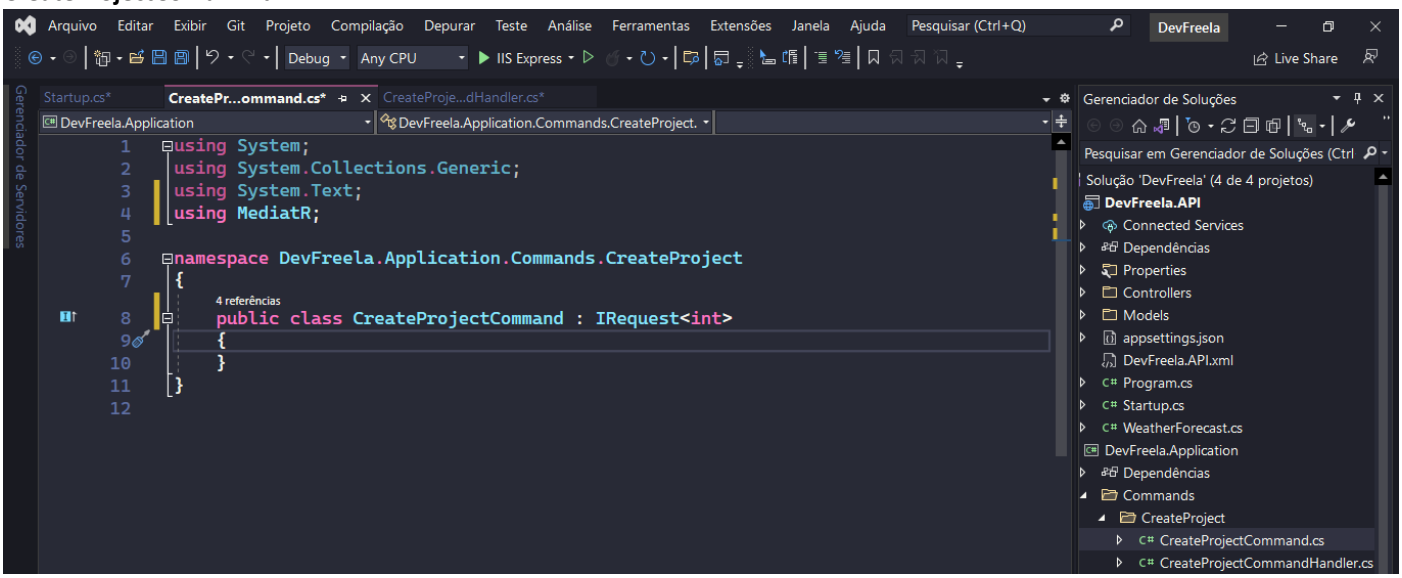
CRIAR PASTAS – Command e Queries em Aplication



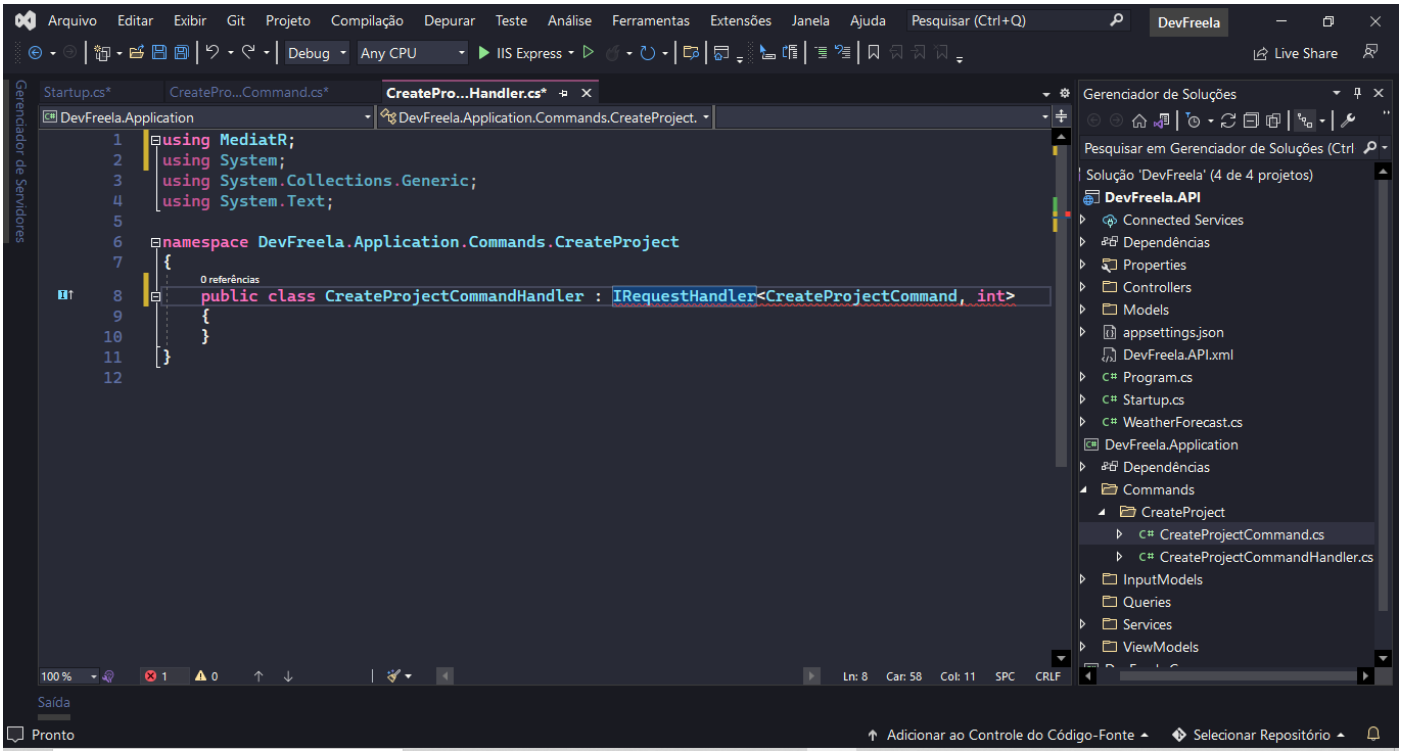
DevFreela.API > Startup (Remover OpenigTime e Exemplas não serão usados) > Adicionar (MediatR.Extensions.Microsoft.DependencyInjection) Para todas camadas que precisarem Startup >



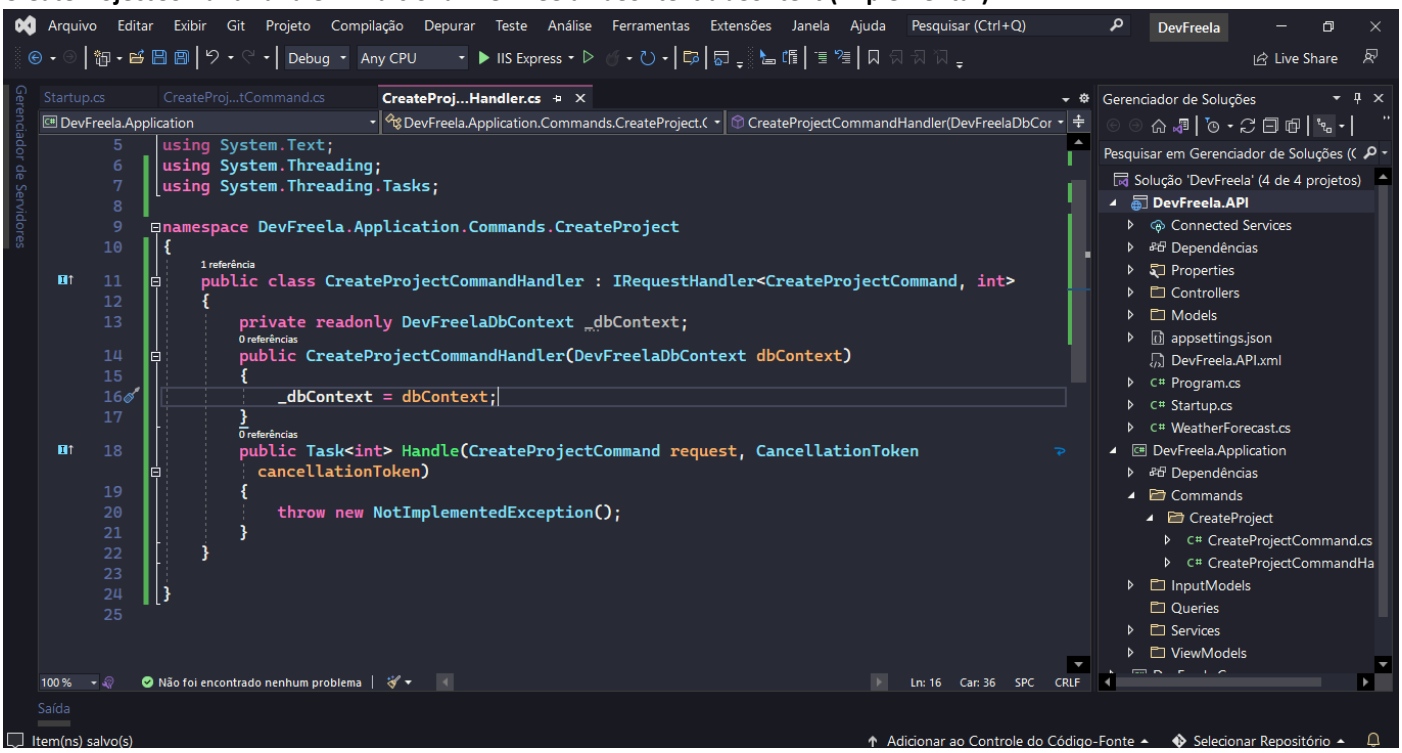
CreateProjectComammd



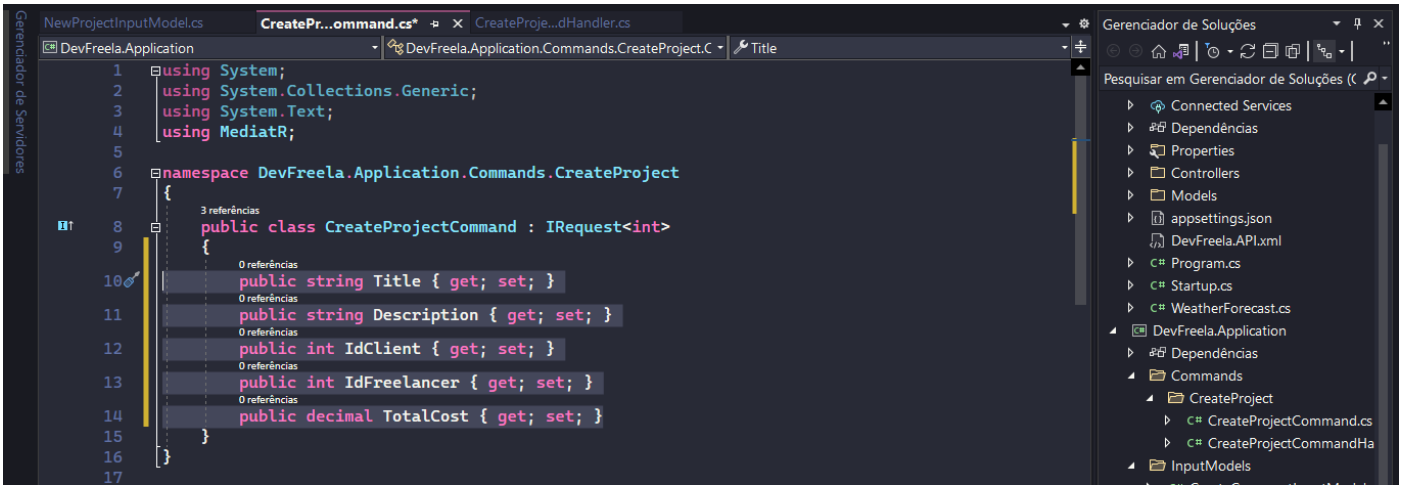
CreatedProjectCommandHandler



CRIANDO OS COMANDS (Verificar as dependências em ProjectService, neste caso DevFreelaDbContext dbContext)
CreateProjectComandHandler > Adicionar DevFreelaDbContext dbContext (Implementar)



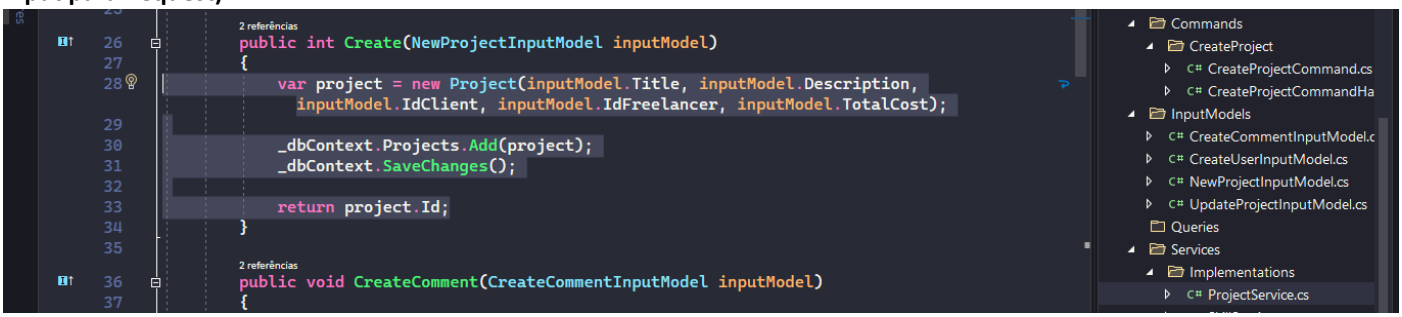
Aplication > InputModel > NewProjectInputModel (Copiar os comandos e colar em Aplication > CreateProjectCommand)



```
using System;
using System.Collections.Generic;
using System.Text;
using MediatR;

namespace DevFreela.Application.Commands.CreateProject
{
    3 referências
    public class CreateProjectCommand : IRequest<int>
    {
        0 referências
        public string Title { get; set; }
        0 referências
        public string Description { get; set; }
        0 referências
        public int IdClient { get; set; }
        0 referências
        public int IdFreelancer { get; set; }
        0 referências
        public decimal TotalCost { get; set; }
    }
}
```

ProjectService (Pego o código que usava para cadastrar um projeto e adiciono CreateProjectCommandHandler > Alterando de Input para Request)



```
2 referências
public int Create(NewProjectInputModel inputModel)
{
    var project = new Project(inputModel.Title, inputModel.Description,
      inputModel.IdClient, inputModel.IdFreelancer, inputModel.TotalCost);

    _dbContext.Projects.Add(project);
    _dbContext.SaveChanges();

    return project.Id;
}

2 referências
public void CreateComment(CreateCommentInputModel inputModel)
{
```

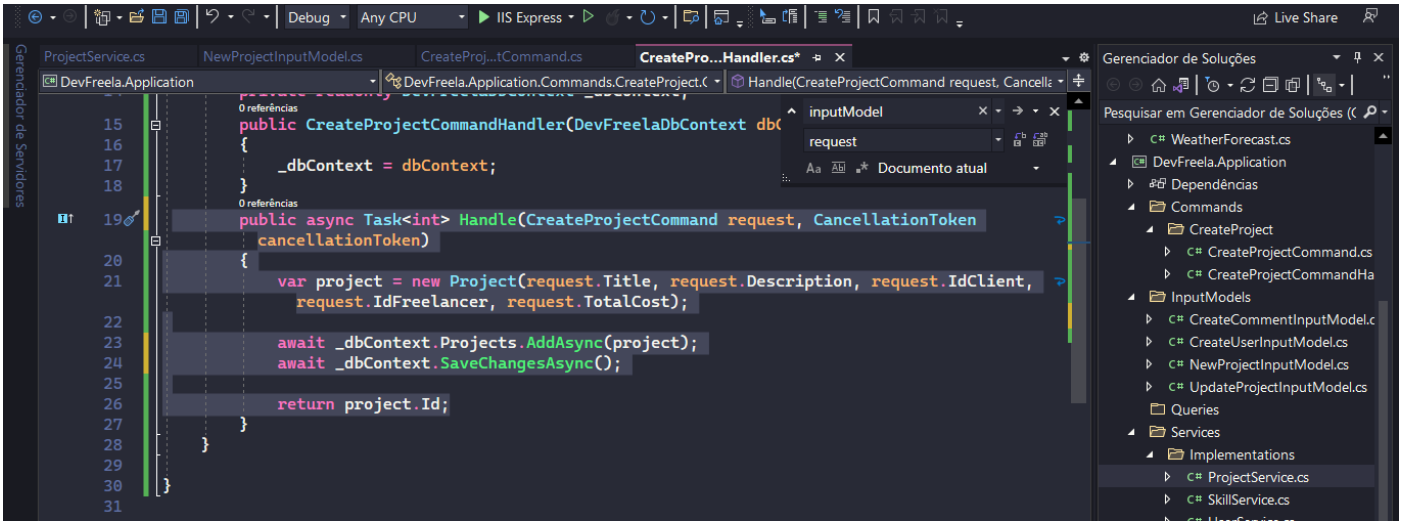
```
0 referências
public CreateProjectCommandHandler(DevFreelaDbContext dbContext)
{
    _dbContext = dbContext;
}
0 referências
public Task<int> Handle(CreateProjectCommand request, CancellationToken
  cancellationToken)
{
    var project = new Project(request.Title, request.Description, request.IdClient,
      request.IdFreelancer, request.TotalCost);

    _dbContext.Projects.Add(project);
    _dbContext.SaveChanges();

    return project.Id;
}
}

}
```

Devemos usar um método ASSINCRONO – ASYNC Quando fazemos uma requisição no banco de dados a Create fica esperando a resposta e fica inativa, quando usamos o Await basicamente delegamos a operação de entrada e saída e a Create fica livre para fazer outras coisas não travando.



```
0 referências
public CreateProjectCommandHandler(DevFreelaDbContext dbC
{
    _dbContext = dbContext;
}
0 referências
public async Task<int> Handle(CreateProjectCommand request, CancellationToken
  cancellationToken)
{
    var project = new Project(request.Title, request.Description, request.IdClient,
      request.IdFreelancer, request.TotalCost);

    await _dbContext.Projects.AddAsync(project);
    await _dbContext.SaveChangesAsync();

    return project.Id;
}
}

}
```

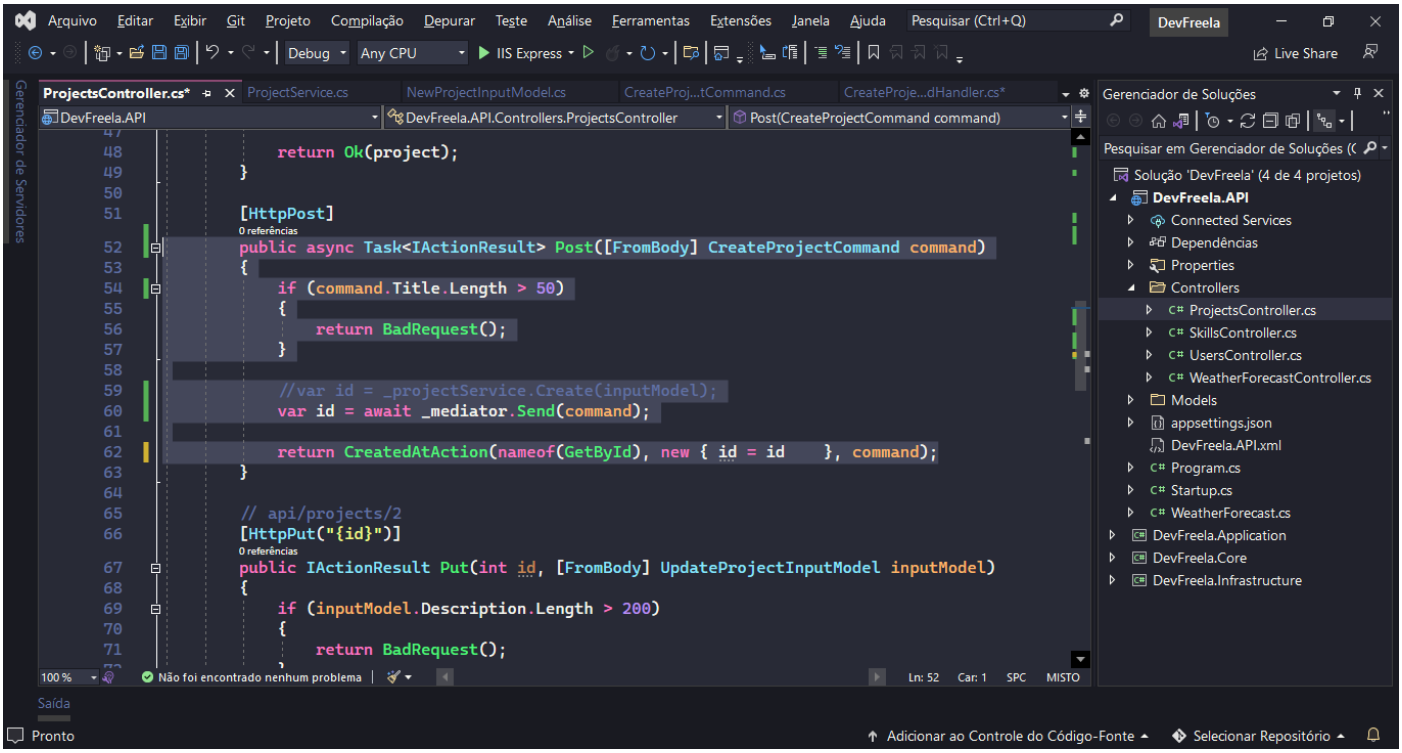
Controller > ProjectController – Aqdicionar o IMediator

```
namespace DevFreela.API.Controllers
{
    [Route("api/projects")]
    1 referência
    public class ProjectsController : ControllerBase
    {
        private readonly IProjectService _projectService;
        private readonly IMediator _mediator;
        0 referências
        public ProjectsController(IProjectService projectService, IMediator
          mediator)
        {
            _projectService = projectService;
            _mediator = mediator;

        }

        // api/projects?query=net core
        [HttpGet]
        0 referências
```


Http Post Project



```
            return Ok(project);
        }

        [HttpPost]
        0 referências
        public async Task<IActionResult> Post([FromBody] CreateProjectCommand command)
        {
            if (command.Title.Length > 50)
            {
                return BadRequest();
            }

            //var id = _projectService.Create(inputModel);
            var id = await _mediator.Send(command);

            return CreatedAtAction(nameof(GetById), new { id = id    }, command);
        }

        // api/projects/2
        [HttpPut("{id}")]
        0 referências
        public IActionResult Put(int id, [FromBody] UpdateProjectInputModel inputModel)
        {
            if (inputModel.Description.Length > 200)
            {
                return BadRequest();
```


Criar pasta CreateComment > Classe CreateCommentComand – Ir em ProjectService > Tipo Void = Unit
Pegar o modelo de entrada F12 em cima CreateCommentInputModel e pegue as entradas

QUERIES – Aplication criar pasta Queries e Classes > Para os View Model – Função do Padrão mediator