Railson Tales de Oliveira

Respostas e evidências do teste técnico

Questões relacionadas a C#

1. Orientação a Objetos:

- Explique o conceito de herança múltipla e como C# aborda esse cenário.

C# não permite heranças múltiplas.

- Explique o polimorfismo em C# e forneça um exemplo prático de como ele pode ser implementado.

Polimorfismo permite que diferentes objetos respondam a mesma mensagem cada um a sua maneira. Por exemplo:

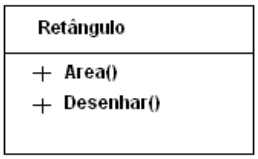
```
public class Shape
{
    // A few example members
    public int X { get; private set; }
    public int Y { get; private set; }
    public int Height { get; set; }
    public int Width { get; set; }

    // Virtual method
    public virtual void Draw()
    {
        Console.WriteLine("Performing base class drawing tasks");
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        // Code to draw a circle...
        Console.WriteLine("Drawing a circle");
        base.Draw();
    }
}
public class Rectangle : Shape
{
    public override void Draw()
    {
        // Code to draw a rectangle...
        Console.WriteLine("Drawing a rectangle");
        base.Draw();
    }
}
```

2. SOLID:

- Descreva o princípio da Responsabilidade Única (SRP) e como ele se aplica em um contexto de desenvolvimento C#.

Uma classe deve fazer apenas uma coisa, deve fazê-la bem e deve fazer somente ela. Se uma classe tem mais de um motivo para ser alterada, ela não segue este princípio.

Vejamos o clássico exemplo da classe Retângulo que possui dois métodos conforme mostrado a seguir:



Métodos:

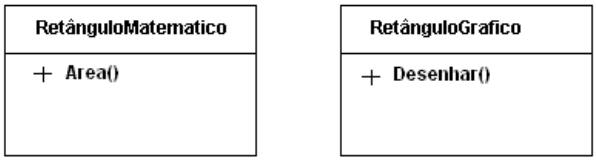
Area() - Calcula a área do Retângulo;

Desenhar() - Desenha o Retângulo;

Este desenho viola o princípio da responsabilidade única - **SRP** pois a classe Retângulo possui duas responsabilidades definidas:

1. **Calcular a área do retângulo usando um modelo matemático;**
2. **Desenhar o retângulo usando uma interface gráfica;**

No caso da classe Retângulo um melhor desenho será separar as duas responsabilidades em duas classes diferentes:



- ⌄Como o princípio da inversão de dependência (DIP) pode ser aplicado em um projeto C# e como isso beneficia a manutenção do código?

O princípio da inversão de dependência diz que devemos depender de abstrações (interfaces) ao invés de implementações (classes concretas) a fim de ter um menor acoplamento entre as camadas do sistema.
A implementação do DIP é relativamente simples, basta criarmos uma interface IPostsService e fazer com que nosso controlador dependa dela e não da implementação concreta (PostsService).

3. Entity Framework (EF):

- ⌄Como o Entity Framework gerencia o mapeamento de objetos para o banco de dados e vice-versa?

O Entity Framework é uma biblioteca ORM que implementa a camada de acesso a dados, baseada em objetos relacionais. Ele permite que você trabalhe com dados do banco de dados como objetos de sua aplicação, sem precisar escrever código SQL para acessá-los. Isso significa que você pode escrever seu código definindo classes e seus relacionamentos, e o Entity Framework se encarrega de traduzir os objetos delas em instruções SQL para o banco de dados.

Ele também é altamente personalizável, permitindo que você configure a maneira como seus dados são mapeados para seus objetos e vice-versa.

Os principais conceitos e componentes do Entity Framework incluem:
Modelo de Dados: É a representação do banco de dados como objetos da sua aplicação. Ele é criado a partir de suas entidades, relacionamentos e propriedades.
Entidade: Classe que representa uma tabela ou uma coleção de dados em seu banco de dados. As entidades são mapeadas para tabelas e suas propriedades são mapeadas para colunas.
No Entity Framework a classe DbSet<T> é utilizada para as operações de acesso a dados de entidades.
Contexto de Dados: Representa a conexão com o banco de dados e é responsável por gerenciar as entidades, realizar operações de banco de dados e persistir as alterações nas entidades. Além disso, também são feitas as configurações das entidades.
LINQ: Linguagem de Consulta Integrada é uma linguagem de consulta de dados que permite que você execute consultas no modelo de dados usando sintaxe de programação.
Migrations: Permite que você gerencie as alterações no banco de dados, incluindo adição, remoção e alteração de tabelas e colunas.
Provedor de Dados: É responsável por fornecer suporte a diferentes tipos de bancos de dados, incluindo SQL Server, MySQL, Cosmos DB, PostgreSQL, SQLite, em memória, e outros.

- Como otimizar consultas no Entity Framework para garantir um desempenho eficiente em grandes conjuntos de dados?

1 - Evite colocar todos os objetos de banco de dados em um único modelo de entidade
2- Desative o controle de alterações (change tracking) para a entidade se isto não for necessário
3 - Use Views geradas previamente para reduzir o tempo de resposta para a primeira solicitação
4 - Evite retornar campos não obrigatórios do banco de dados.
5 - Escolha a coleção apropridada para a manipulação de dados
6 - Utilize consultas compiladas sempre que necessário
7 - Retorne somente o número de registros requeridos na paginação
8 - Evite usar o método Contains
9- Evita utilizar Views
10 - Debug e otimize as consultas LINQ (LINQPad)

4. WebSockets:

- Explique o papel dos WebSockets em uma aplicação C# e como eles se comparam às solicitações HTTP tradicionais.

WebSockets é uma tecnologia que permite mantermos clientes conectados aos nossos servidores e trafegar mensagens mais rápidas. Uma requisição HTTP tem alguns passos a serem seguidos, dentre eles o handshake (Aperto de mão) e abertura da conexão. Somente depois destes passos trafegamos os dados, e em seguida a conexão é fechada. Neste modelo mais convencional, precisamos realizar estes passos a cada requisição ao servidor, enquanto nos WebSockets, uma vez conectados, trafegamos apenas os dados.

- Quais são as principais considerações de segurança ao implementar uma comunicação baseada em WebSockets em uma aplicação C#?

Compartilhamento de recurso entre origens (CORS)
Restrição de origem do WebSocket
ConnectionId
Log de token de acesso
Exceções
Gerenciamento de buffer

5. Arquitetura:

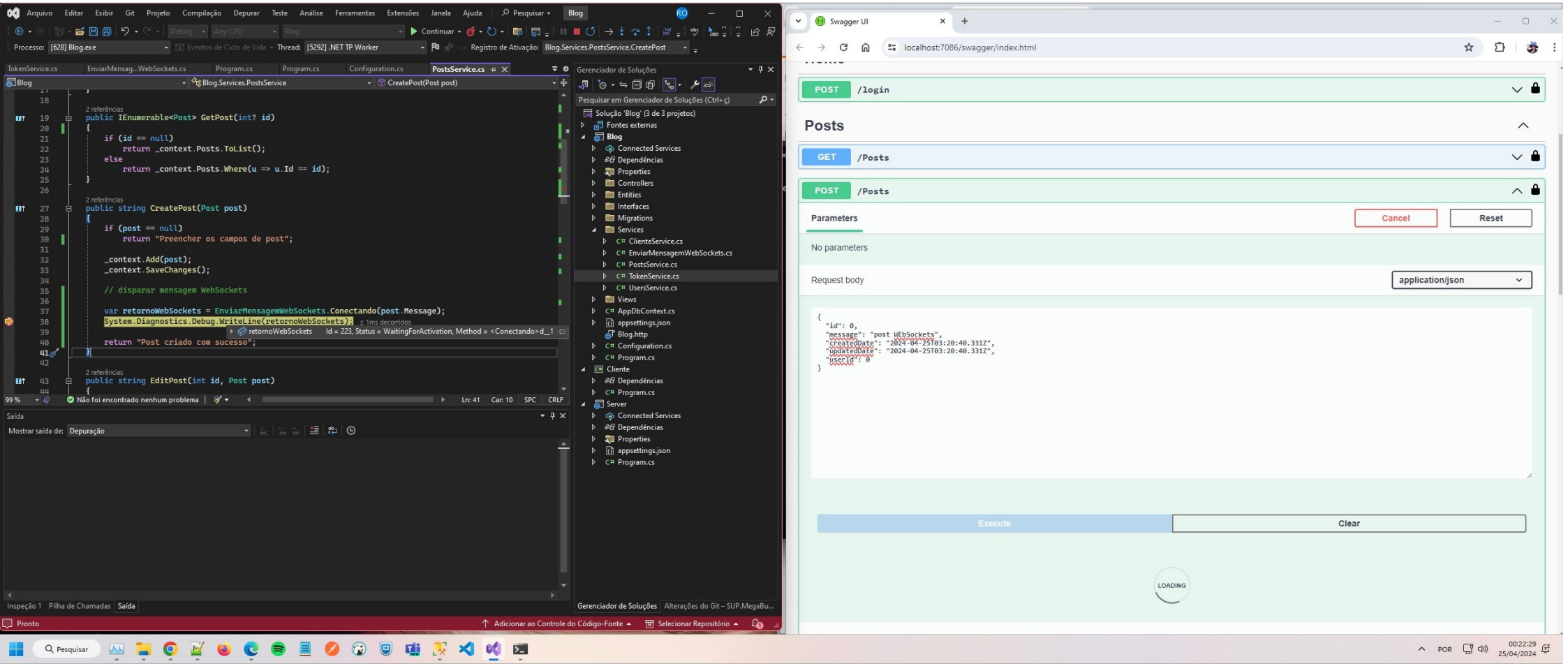
- Descreva a diferença entre arquitetura monolítica e arquitetura de microsserviços. Qual seria sua escolha ao projetar uma aplicação C#?

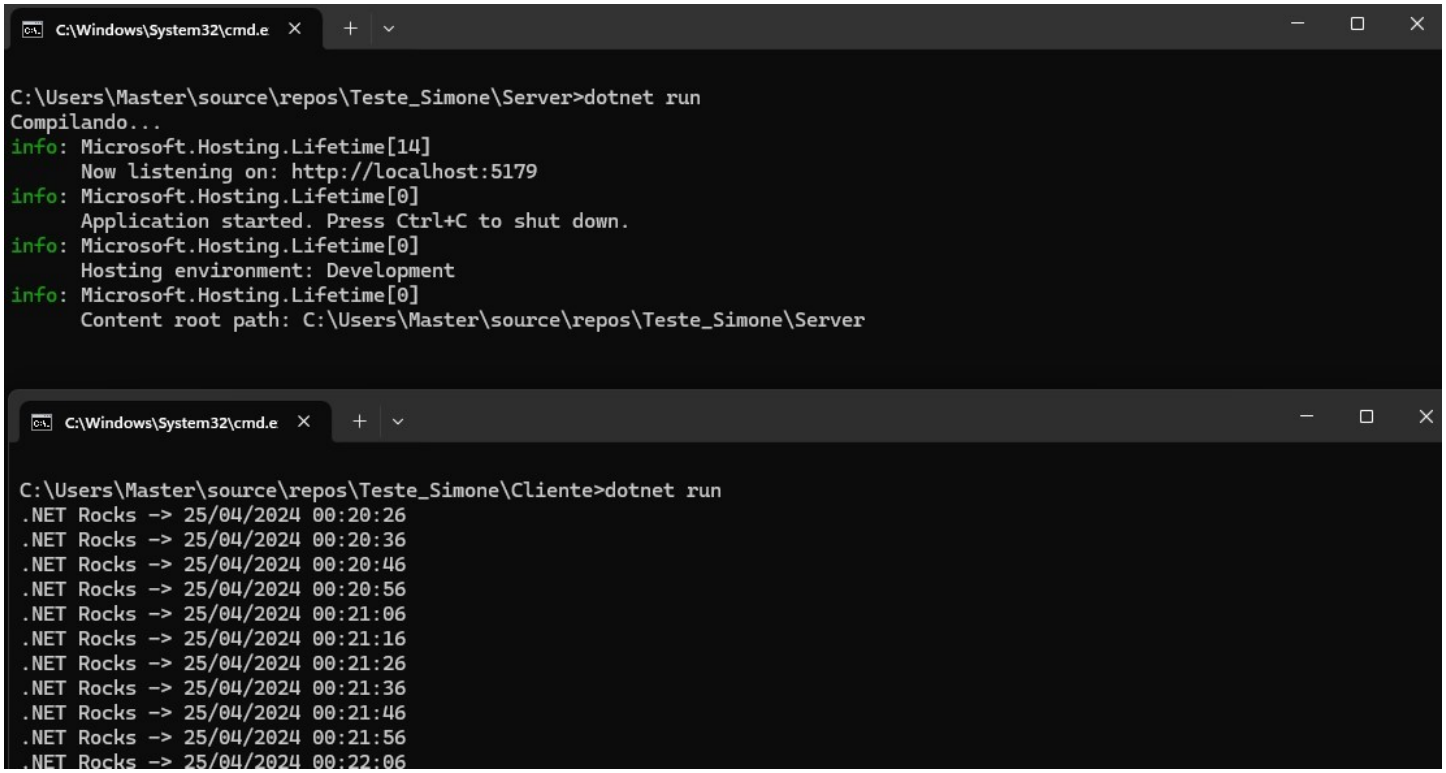
Uma arquitetura monolítica é um modelo tradicional de desenvolvimento de software que usa uma base de código para executar várias funções comerciais. Todos os componentes de software em um sistema monolítico são interdependentes devido aos mecanismos de troca de dados dentro do sistema. É restritivo e demorado modificar a arquitetura monolítica, pois pequenas mudanças afetam grandes áreas da base de código. Em contraste, os microsserviços são uma abordagem arquitetônica que compõe o software em pequenos componentes ou serviços independentes. Cada serviço executa uma única função e se comunica com outros serviços por meio de uma interface bem definida. Como eles são executados de forma independente, você pode atualizar, modificar, implantar ou escalar cada serviço conforme necessário.
Na escolha de projetar uma aplicação C# precisa levar várias questões antes como Tamanho da aplicação, Competência da equipe e Infraestrutura.

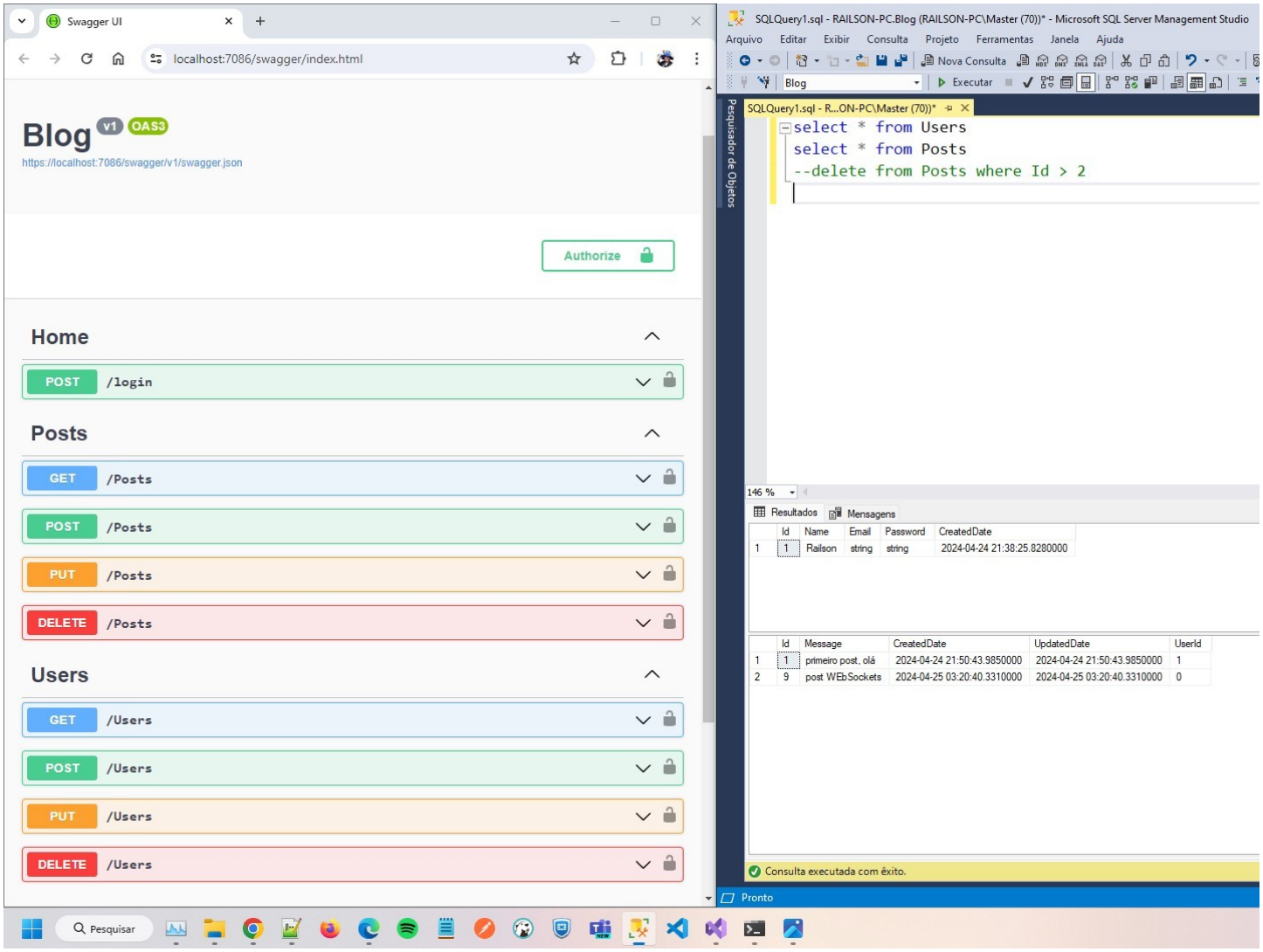
- Como você escolheria entre a arquitetura de microsserviços e a arquitetura monolítica ao projetar uma aplicação C# que precisa ser altamente escalável?

Escolheria a arquitetura de microsserviços, pois seria possível hospedar os serviços em nuvem para ajudar a garantir escalabilidade, tolerância a falhas e alta disponibilidade. Quando se separa a aplicação monolítica para microsserviços fica mais fácil escalar as pequenas partes separadas que toda a aplicação junta num só.

Teste prático C#
Projeto de Blog Simples - Evidências


Arquivo Editar Exibir Git Projeto Compilação Depurar Teste Análise Ferramentas Extensões Janela Ajuda
Blog
Continuar
Processo: [628] Blog.exe
Thread: [5292] .NET TP Worker
TokenService.cs
EnviarMensag...WebSockets.cs
Program.cs
Configuration.cs
PostsService.cs
Blog.Services.PostsService
CreatePost(Post post)
2 referências
public IEnumerable<Post> GetPost(int? id)
{
if (id == null)
return _context.Posts.ToList();
else
return _context.Posts.Where(u => u.Id == id);
}
2 referências
public string CreatePost(Post post)
{
if (post == null)
return "Preencher os campos de post";
_context.Add(post);
_context.SaveChanges();
// disparar mensagem WebSockets
var retornoWebSockets = EnviarMensagemWebSockets.Conectando(post.Message);
System.Diagnostics.Debug.WriteLine(retornoWebSockets);
retornoWebSockets Id = 223, Status = WaitingForActivation, Method = <Conectando>d__1
return "Post criado com sucesso";
}
2 referências
public string EditPost(int id, Post post)
{
Não foi encontrado nenhum problema
Ln: 41 Car: 10 SPC CRLF
Saída
Mostrar saída de: Depuração
Inspeção 1 Pilha de Chamadas Saída
Gerenciador de Soluções
Pesquisar em Gerenciador de Soluções (Ctrl+ç)
Solução 'Blog' (3 de 3 projetos)
Fontes externas
Blog
Connected Services
Dependências
Properties
Controllers
Entities
Interfaces
Migrations
Services
ClienteService.cs
EnviarMensagemWebSockets.cs
PostsService.cs
TokenService.cs
UsersService.cs
Views
AppDbContext.cs
appsettings.json
Blog.http
Configuration.cs
Program.cs
Cliente
Dependências
Program.cs
Server
Connected Services
Dependências
Properties
appsettings.json
Program.cs
Gerenciador de Soluções
Alterações do Git – SUP.MegaBu...
Pronto
Adicionar ao Controle do Código-Fonte
Selecionar Repositório
Swagger UI
localhost:7086/swagger/index.html
POST /login
Posts
GET /Posts
POST /Posts
Parameters
Cancel
Reset
No parameters
Request body
application/json
{
"id": 0,
"message": "post WEbSockets",
"createdDate": "2024-04-25T03:20:40.331Z",
"updatedDate": "2024-04-25T03:20:40.331Z",
"userId": 0
}
Execute
Clear
LOADING
Pesquisar
POR
00:22:29
25/04/2024


C:\Windows\System32\cmd.e
C:\Users\Master\source\repos\Teste_Simone\Server>dotnet run
Compilando...
info: Microsoft.Hosting.Lifetime[14]
Now listening on: http://localhost:5179
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:\Users\Master\source\repos\Teste_Simone\Server
C:\Windows\System32\cmd.e
C:\Users\Master\source\repos\Teste_Simone\Cliente>dotnet run
.NET Rocks -> 25/04/2024 00:20:26
.NET Rocks -> 25/04/2024 00:20:36
.NET Rocks -> 25/04/2024 00:20:46
.NET Rocks -> 25/04/2024 00:20:56
.NET Rocks -> 25/04/2024 00:21:06
.NET Rocks -> 25/04/2024 00:21:16
.NET Rocks -> 25/04/2024 00:21:26
.NET Rocks -> 25/04/2024 00:21:36
.NET Rocks -> 25/04/2024 00:21:46
.NET Rocks -> 25/04/2024 00:21:56
.NET Rocks -> 25/04/2024 00:22:06


Swagger UI
localhost:7086/swagger/index.html
Blog
v1
OAS3
https://localhost:7086/swagger/v1/swagger.json
Authorize
Home
POST /login
Posts
GET /Posts
POST /Posts
PUT /Posts
DELETE /Posts
Users
GET /Users
POST /Users
PUT /Users
DELETE /Users
SQLQuery1.sql - RAILSON-PC.Blog (RAILSON-PC\Master (70))* - Microsoft SQL Server Management Studio
Arquivo Editar Exibir Consulta Projeto Ferramentas Janela Ajuda
Nova Consulta
Executar
Pesquisador de Objetos
SQLQuery1.sql - R...ON-PC\Master (70))*
select * from Users
select * from Posts
--delete from Posts where Id > 2
146 %
Resultados
Mensagens
Id Name Email Password CreatedDate
1 1 Railson string string 2024-04-24 21:38:25.8280000
Id Message CreatedDate UpdatedDate UserId
1 1 primeiro post, olá 2024-04-24 21:50:43.9850000 2024-04-24 21:50:43.9850000 1
2 9 post WEbSockets 2024-04-25 03:20:40.3310000 2024-04-25 03:20:40.3310000 0
Consulta executada com êxito.
Pronto
Pesquisar