

# Gerenciamento de Biblioteca

---

DOCUMENTAÇÃO

Guilherme dos Santos

## Sumário

1. Visão Geral do Projeto .....	Página
2. Configuração do Ambiente .....	Página
3. Detalhamento das Funcionalidades .....	Página
4. Testes Automatizados .....	Página
5. Considerações Finais .....	Página
6. Referências .....	Página

## **1. Visão Geral do Projeto**

### **1.1 Nome do Projeto**

Gerenciamento de Biblioteca

### **1.2 Descrição**

Este projeto permite o cadastro, busca e listagem de livros e autores. Inclui testes automatizados para garantir o funcionamento correto.

### **1.3 Tecnologias Utilizadas**

Linguagem: C#

IDE: Visual Studio

Banco de Dados: SQL Server (produção) PostgreSQL (desenvolvimento)

Gerenciador de BD: DBeaver

Testes Automatizados: Selenium

## **2. Configuração do Ambiente**

### **2.1 Requisitos de Sistema**

- Windows 10 ou superior
- Visual Studio 2019 ou superior
- SQL Server 2019
- PostgreSQL 13
- DBeaver 21.0 ou superior
- Selenium WebDriver

### **2.2 Passos para Configuração**

#### **2.2.1 Instalar Visual Studio**

- Baixe e instale o Visual Studio.
- Selecione as workloads de desenvolvimento em C# e .NET.

#### **2.2.2 Instalar SQL Server e PostgreSQL**

- Baixe e instale o SQL Server.
- Baixe e instale o PostgreSQL.

#### **2.2.3 Instalar DBeaver**

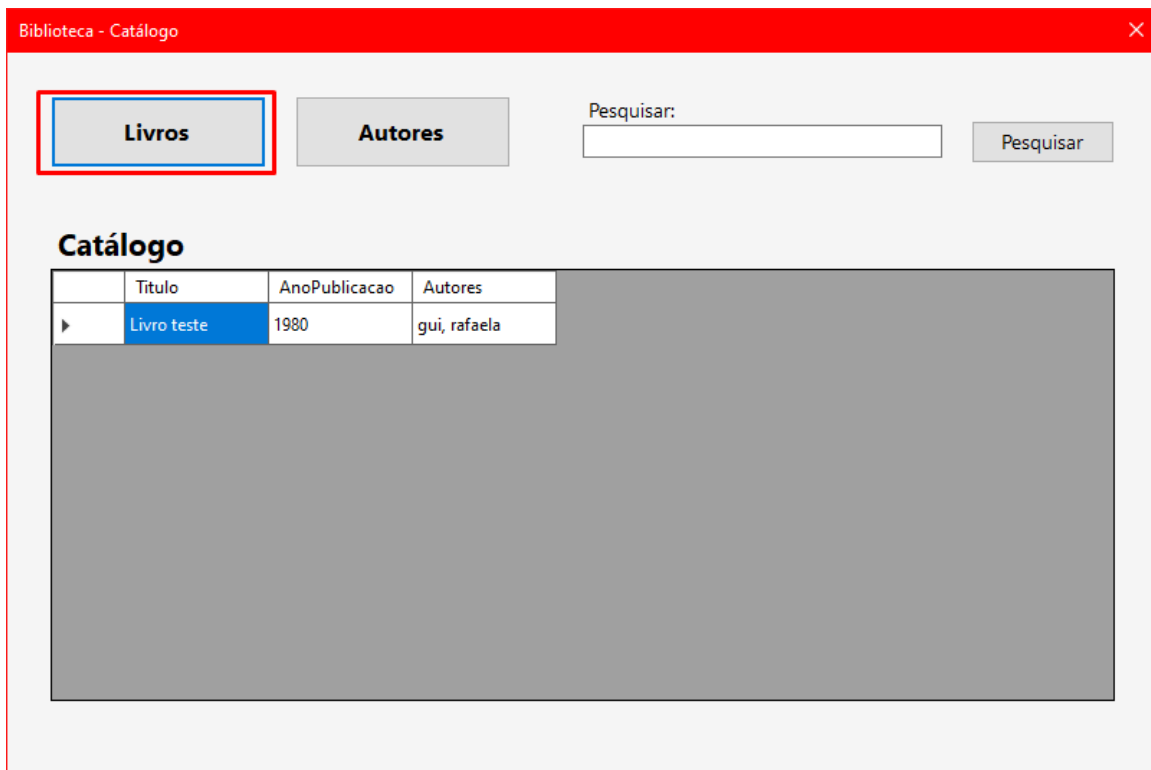
- Baixe e instale o DBeaver.

## **3. Detalhamento das Funcionalidades**

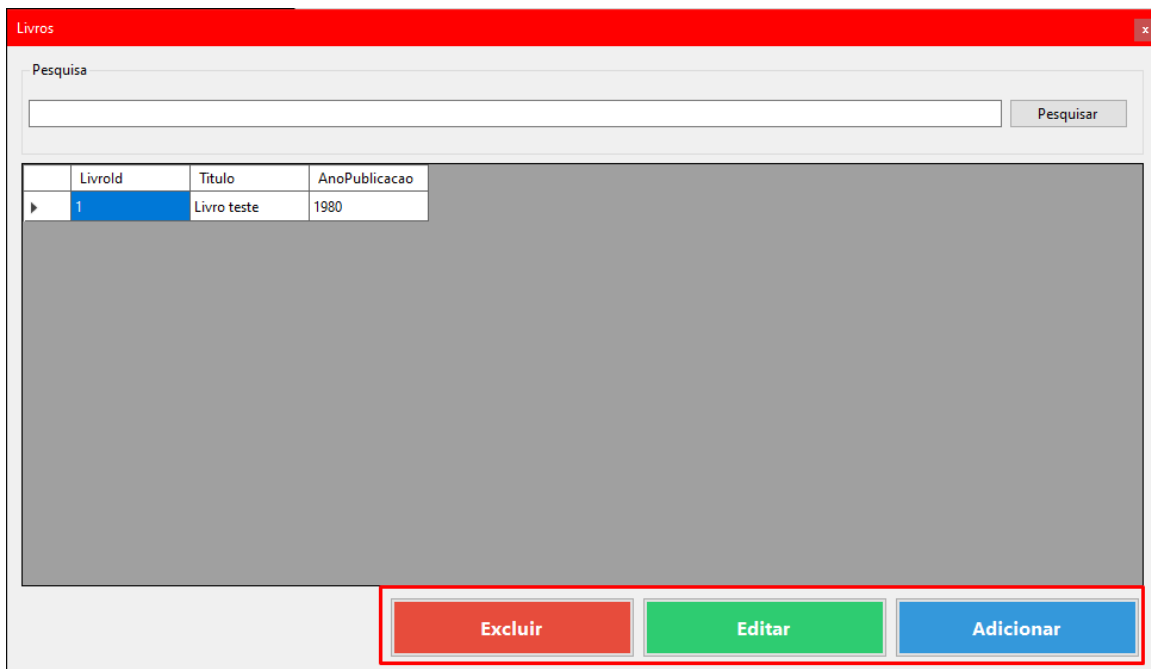
### **3.1 Cadastro de Livros**

O sistema permite excluir, editar e o cadastro de novos livros, especificando título, ano de publicação e autores associados.

Primeiro passo, entrar no formulário de livros:



Segundo passo, escolher a ação que deseja realizar referente aos livros:



Se a opção editar ou adicionar for escolhida, abaixo está um exemplo do formulário completo:

Detalhes

## Detalhes do Livro

Informações do Livro

Título

Livro teste

Ano da  
Publicação

1980

Autores

☒ gui

☒ rafaela

Salvar

Cancelar

### 3.2 Cadastro de Autores

O sistema permite excluir, editar cadastro de novos autores, especificando nome e data de nascimento.

Primeiro passo, entrar no formulário de Autores:

Biblioteca - Catálogo

**Livros** **Autores**

Pesquisar:  **Pesquisar**

**Catálogo**

	Titulo	AnoPublicacao	Autores
▶	Livro teste	1980	gui, rafaela

Segundo passo, escolher a ação que deseja realizar referente aos autores:

**Autores**

Pesquisa

**Pesquisar**

	AutorId	Nome	DataNascimento
▶	2	gui	12/12/2003
	3	rafaela	12/12/2000

**Excluir** **Editar** **Adicionar**

Se a opção editar ou adicionar for escolhida, abaixo está um exemplo do formulário completo:

Detalhes

x

## Detalhes do Autor

Informações do Autor

Nome

gui

Data de Nascimento

12/12/2003

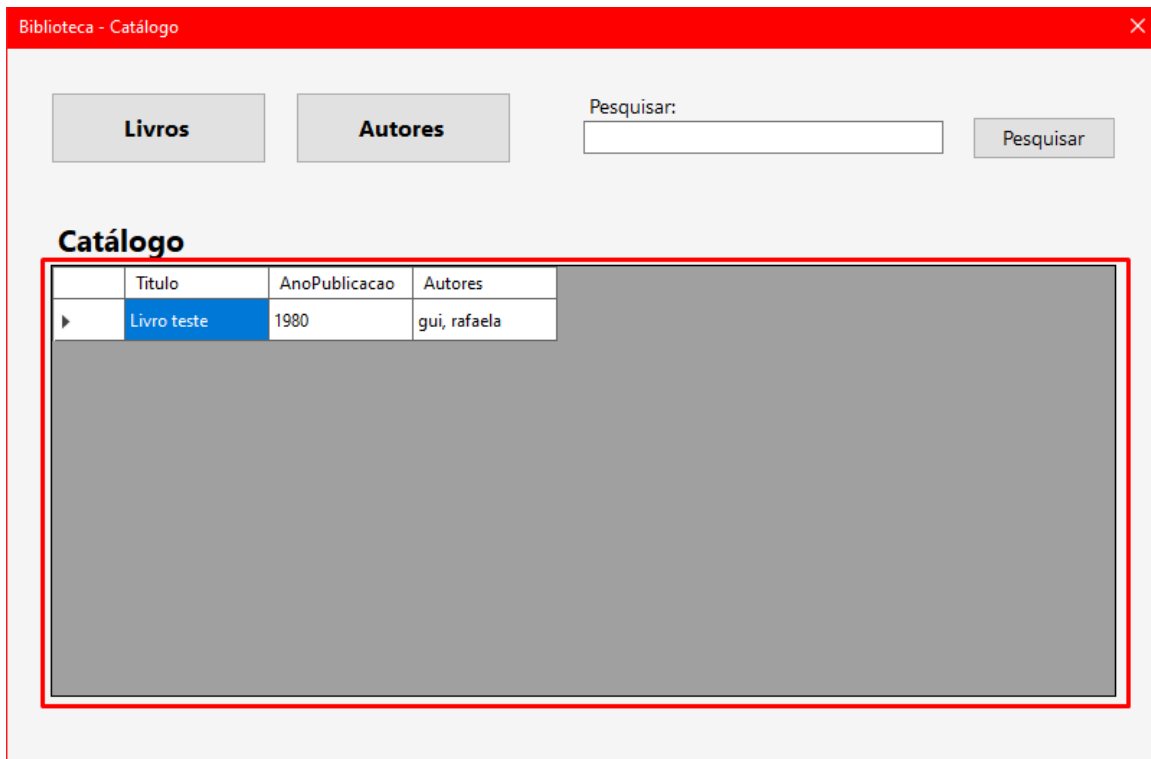
Salvar

Cancelar

### 3.3 Relacionamento entre Livros e Autores

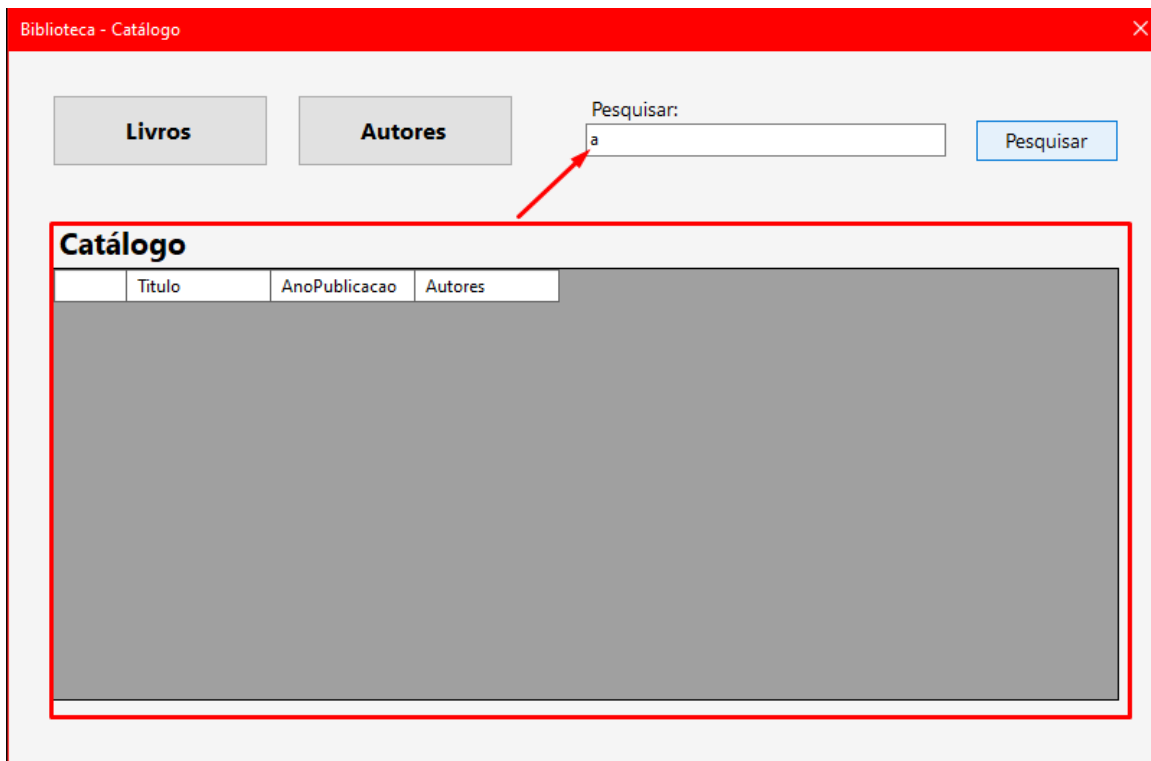
Utiliza-se a tabela de junção AutorLivro para associar livros e autores.





### 3.4 Listagem e Pesquisa

O sistema permite listar e pesquisar livros e autores cadastrados.



Livros

Autores

Pesquisar:

L

Pesquisar

**Catálogo**

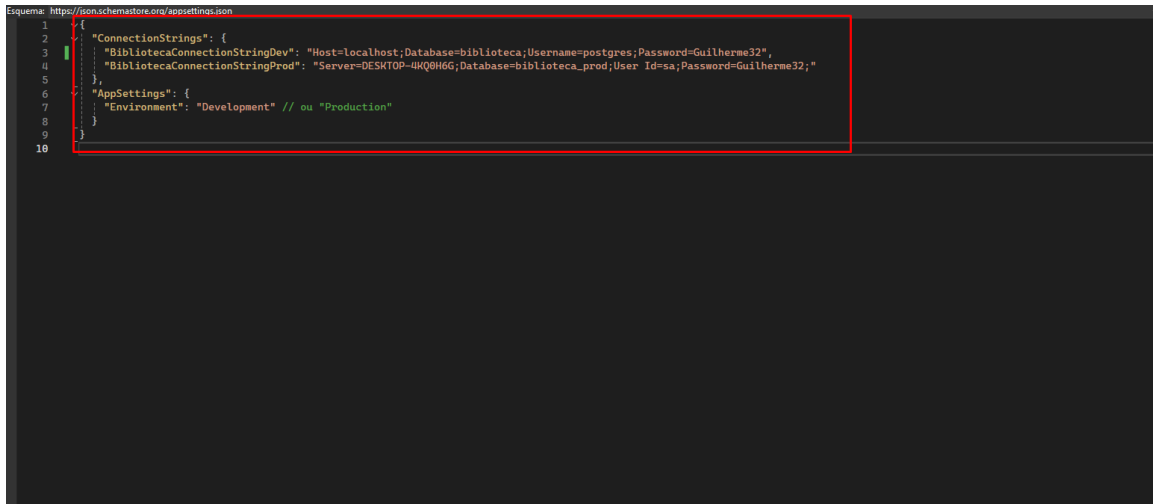
	Titulo	AnoPublicacao	Autores
▶	Livro teste	1980	gui, rafaela

## 4. Persistência de Dados

### 4.1 Configuração do Entity Framework

#### 4.1.1 Criação do Banco de Dados

- Use o Entity Framework Core para gerenciar a criação e migração do banco de dados.
- Configure as conexões de banco de dados no arquivo appsettings.json.



```
1 {
2   "ConnectionStrings": {
3     "BibliotecaConnectionStringDev": "Host=localhost;Database=biblioteca;Username=postgres;Password=Guilherme32",
4     "BibliotecaConnectionStringProd": "Server=DESKTOP-4KQ6H6G;Database=biblioteca_prod;User Id=sa;Password=Guilherme32;"
5   },
6   "AppSettings": {
7     "Environment": "Development" // ou "Production"
8   }
9 }
10
```

### 4.2 Adicionar Migrations e Atualizar o Banco de Dados

No Package Manager Console, execute os seguintes comandos:

Add-Migration NomeDaMigration

Update-Database

CASO NECESSÁRIO O SCRIPT SQL ESTARÁ NO REPOSITÓRIO

## 6. Modelagem de Dados

A modelagem de dados do projeto é baseada em três classes principais: Autor, Livro e AutorLivro, que estabelecem o relacionamento entre autores e livros.

### 6.1 Autor

Representa um autor na biblioteca.

Propriedades:

AutorId (int): Identificador único do autor.

Nome (string): Nome do autor.

DataNascimento (DateTime): Data de nascimento do autor.

AutorLivros (List<AutorLivro>): Lista de objetos AutorLivro que representam a relação muitos-para-muitos entre autores e livros.

```
namespace BibliotecaApp.Models
{
    9 referências
    public class Autor
    {
        2 referências
        public int AutorId { get; set; }
        5 referências
        public string Nome { get; set; }
        3 referências
        public DateTime DataNascimento { get; set; }
        1 referência
        public List<AutorLivro> AutorLivros { get; set; } = new List<AutorLivro>();
    }
}
```

## 6.2 Livro

Representa um livro na biblioteca.

Propriedades:

LivroId (int): Identificador único do livro.

Titulo (string): Título do livro.

AnoPublicacao (int): Ano de publicação do livro.

AutorLivros (List<AutorLivro>): Lista de objetos AutorLivro que representam a relação muitos-para-muitos entre livros e autores.

```
namespace BibliotecaApp.Models
{
    10 referências
    public class Livro
    {
        4 referências
        public int LivroId { get; set; } // Ajustei o nome da propriedade para LivroId
        6 referências
        public string Titulo { get; set; }
        4 referências
        public int AnoPublicacao { get; set; }
        6 referências
        public List<AutorLivro> AutorLivros { get; set; } = new List<AutorLivro>();
    }
}
```

### 6.3 AutorLivro

Classe de junção que representa a relação muitos-para-muitos entre Autor e Livro.

Propriedades:

AutorId (int): Identificador do autor.

Autor (Autor): Instância da classe Autor.

LivroId (int): Identificador do livro.

Livro (Livro): Instância da classe Livro.

```
namespace BibliotecaApp.Models
{
    10 referências
    public class AutorLivro
    {
        5 referências
        public int AutorId { get; set; }
        5 referências
        public Autor Autor { get; set; }
        6 referências
        public int LivroId { get; set; }
        1 referência
        public Livro Livro { get; set; }
    }
}
```

## 7. Testes Automatizados

### 7.1 Ferramentas Utilizadas

- Selenium WebDriver para automação dos testes de interface.

### 7.2 Explicação de como foi realizado os testes

Como o sistema foi desenvolvido com Windows Form, o Selenium não é eficaz para esse tipo de abordagem, então foi desenvolvido uma interface web simples, com o BackEnd sendo uma copia exata do sistema Desktop, para nos mantermos dentro das normas!

### 7.3 Teste cadastrar novo livro

- Navega até a URL da aplicação.
- Espera até 10 segundos para que o campo de entrada do título esteja presente na página.
- Preenche o título do livro e o ano de publicação.
- Clica no botão de salvar.
- Espera até que a lista de livros seja atualizada e contenha o novo livro cadastrado.
- Verifica se o livro foi cadastrado corretamente ao buscar o livro na lista de livros exibida na página.

```
[TestMethod]
public void TesteCadastroNovoLivro()
{
    driver.Navigate().GoToUrl(URL);

    // Espera até 10 segundos para que o elemento "Título" esteja presente na página
    WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
    var txtTitulo = wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementIsVisible(By.Id("Titulo")));

    // Preenche o título do livro
    txtTitulo.SendKeys("Livro Teste");

    // Preenche o ano de publicação
    driver.FindElement(By.Id("AnoPublicacao")).SendKeys("2023");

    // Clic no botão de salvar
    driver.FindElement(By.Id("btnSalvar")).Click();

    // Espera até que a lista de livros seja atualizada e contenha o novo livro
    var livrosList = wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementIsVisible(By.Id("livrosList")));
    var livroCadastrado = livrosList.FindElement(By.XPath("//li[contains(text(), 'Livro Teste (Ano: 2023)')]")).Text;

    // Verifica se o livro foi cadastrado corretamente
    Assert.AreEqual("Livro Teste (Ano: 2023)", livroCadastrado);
}
```

### 7.4 Teste Buscar Livro Cadastrado

- Navega até a URL da aplicação.
- Simula a busca por um livro já cadastrado preenchendo o campo de busca e clicando no botão de pesquisa.
- Espera até 10 segundos para que a lista de livros seja atualizada e contenha o livro buscado.
- Verifica se o livro buscado está presente na lista de resultados exibida na página, comparando o texto do livro encontrado com o espera

```

[TestMethod]
0 | 0 referências
public void TesteBuscaLivroCadastrado()
{
    driver.Navigate().GoToUrl(URL);

    // Simula a busca por um livro cadastrado
    driver.FindElement(By.Id("IdBusca")).SendKeys("Livro Teste");
    driver.FindElement(By.Id("btnPesquisar")).Click();

    // Espera até que a lista de livros seja atualizada e contenha o livro buscado
    WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
    var livrosList = wait.Until(ExpectedConditions.ElementIsVisible(By.Id("livrosList")));

    // Verifica se o livro buscado está na lista de resultados
    var livroBuscado = livrosList.FindElement(By.XPath("//li[contains(text(), 'Livro Teste (Ano: 2023)')]"));
    Assert.AreEqual("Livro Teste (Ano: 2023)", livroBuscado);
}

```

## 7.5 Teste Listagem de Livros

-Navega até a URL da aplicação.

-Espera até 10 segundos para que a lista de livros esteja visível na página.

-Verifica se a lista de livros contém pelo menos um item, garantindo que a listagem de livros não está vazia.

```

[TestMethod]
0 | 0 referências
public void TesteListagemLivros()
{
    driver.Navigate().GoToUrl(URL);

    WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
    var livrosList = wait.Until(ExpectedConditions.ElementIsVisible(By.Id("livrosList")));

    // Verifica se a list de livros contém pelo menos um item
    var livros = livrosList.FindElements(By.TagName("li"));
    Assert.IsTrue(livros.Count > 0, "A lista de livros está vazia.");
}

```

## 7.6 Instruções para realizar os testes

- Clone o repositório da nossa cópia web simplificada:  
<https://github.com/JabuS2/BibliotecaWeb-TesteAutomatizado>

- Configure a String de conexão da mesma forma que foi mostrado anteriormente(4.1.1)

-Instale as dependências, da mesma forma que foi mostrado anteriormente!

-Inicie a aplicação

-Clone o repositório do nosso teste unitário: <https://github.com/JabuS2/Selenium-Automatizado>

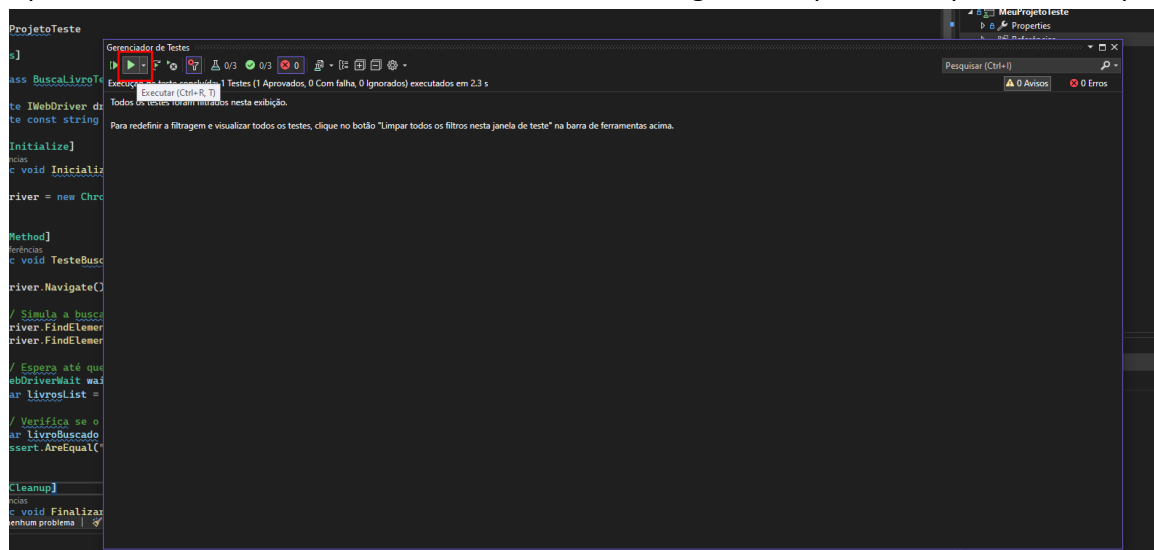
-Configure a String de conexão da mesma forma que foi mostrado anteriormente(4.1.1)

-Instale as dependências, da mesma forma que foi mostrado anteriormente!

-Configure a porta correta:

```
private const string URL = "http://localhost:5189";
```

Aperte Ctrl + E + T, logo após aperte aqui:



## 8. Instruções de Execução

### 8.1 Clone o repositório

- Entre no link: <https://github.com/JabuS2/BibliotecaSanchez>

- Clone o repositório

### 8.2 Configuração do projeto

- Abra o projeto no Visual Studio.

- Configure as strings de conexão para os bancos de dados SQL Server e PostgreSQL.

### 8.3 Execução da Aplicação

- Compile e execute o projeto a partir do Visual Studio.



## 9. Instruções de Implementação no Cliente

Para garantir uma implementação eficiente do sistema de gerenciamento de biblioteca no ambiente do cliente, siga as instruções abaixo para criar um executável e instalar o sistema.

### 9.1 Requisitos de Sistema

Sistema Operacional: Windows 10 ou superior

Software Necessário:

- .NET Runtime (se não incluído no executável)
- SQL Server (opcional, se usado no backend)
- PostgreSQL (opcional, se usado no backend)

### 9.2 Criando um Executável

#### 1. Preparar o Projeto para Publicação

- Abra o projeto no Visual Studio.
- Certifique-se de que todas as dependências estejam instaladas e o projeto esteja funcionando corretamente.

#### 2. Publicar o Projeto

- No Visual Studio, clique com o botão direito no projeto na Solution Explorer.
- Selecione 'Publish'.
- Configure um novo perfil de publicação:
  - Target: Folder
  - Configuration: Release
  - Target Runtime: 'win-x64'
- Clique em 'Publish' para gerar os arquivos de publicação.

#### 3. Criar um Instalador (Opcional)

Para criar um instalador que inclua o executável e qualquer outra dependência necessária:

- Utilize uma ferramenta como o Inno Setup ou WiX Toolset.

Instruções para Inno Setup:

- Baixe e instale o Inno Setup.

- Crie um novo script de instalação e configure-o para incluir os arquivos publicados do projeto.
- Compile o script para gerar um instalador `.exe`.

### 9.3 Instruções de Instalação no Cliente

#### 1. Pré-Requisitos

- Certifique-se de que o .NET Runtime esteja instalado no computador do cliente. Caso contrário, baixe e instale-o a partir do site oficial da Microsoft.

#### 2. Instalação do Banco de Dados

Se o sistema utiliza um banco de dados SQL Server ou PostgreSQL, siga as instruções abaixo:

- SQL Server:
  - Baixe e instale o SQL Server a partir do site oficial.
  - Configure o SQL Server conforme necessário e crie o banco de dados utilizando os scripts SQL fornecidos.
- PostgreSQL:
  - Baixe e instale o PostgreSQL a partir do site oficial.
  - Configure o PostgreSQL conforme necessário e crie o banco de dados utilizando os scripts SQL fornecidos.

#### 3. Instalação do Sistema

- Execute o instalador `.exe` gerado pelo Inno Setup ou WiX Toolset.
- Siga as instruções do assistente de instalação para concluir a instalação do sistema.

#### 4. Configuração do Sistema

- Após a instalação, abra o arquivo de configuração (`appsettings.json` ou outro) no diretório de instalação.
- Configure a string de conexão para o banco de dados utilizado.
- Salve as alterações no arquivo de configuração.

#### 5. Executar o Sistema

- Navegue até o diretório de instalação.
- Execute o arquivo principal do sistema.

- Acesse a interface do usuário para começar a usar o sistema de gerenciamento de biblioteca.