

Aplicativo MVC – Parte 1

José Cassiano Grassi Gunji

Neste tutorial vamos desenvolver um aplicativo usando a arquitetura MVC com duas interfaces gráficas, uma desktop e outra web. Para tanto, vamos usar a tecnologia .Net. Nesta primeira parte vamos criar a infraestrutura de back-office e a interface gráfica de modalidade Web. Na Parte 2 vamos completar o aplicativo com a interface gráfica em modalidade desktop.

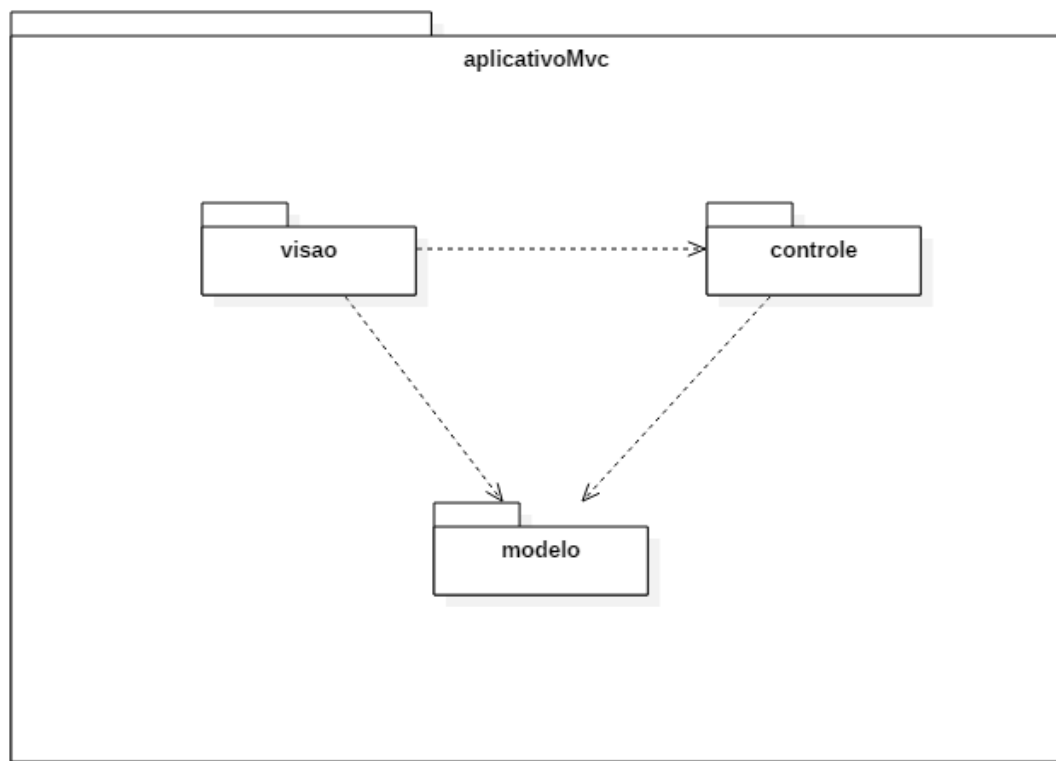
Este aplicativo será uma simples agenda de telefones. Suas funcionalidades podem ser representadas pelo diagrama de casos de uso a seguir.

Figura 1: Diagrama de casos de uso do aplicativo MVC.



Pelo diagrama, vemos que um usuário tem acesso às funcionalidades de acesso a dados (CRUD) relacionadas à manipulação de dados de pessoas de sua agenda de telefones. Esta agenda será armazenada em um banco de dados. Para tornar este aplicativo mais versátil, podemos fazer com que ele seja compatível com vários gerenciadores de banco de dados, por exemplo, MySql, SqlServer, MariaDB, Oracle, SQLite, DB2 e assim por diante. Veremos que com o uso do MVC, esta versatilidade pode ser feita facilmente.

Figura 2: Diagrama de pacotes da arquitetura MVC.

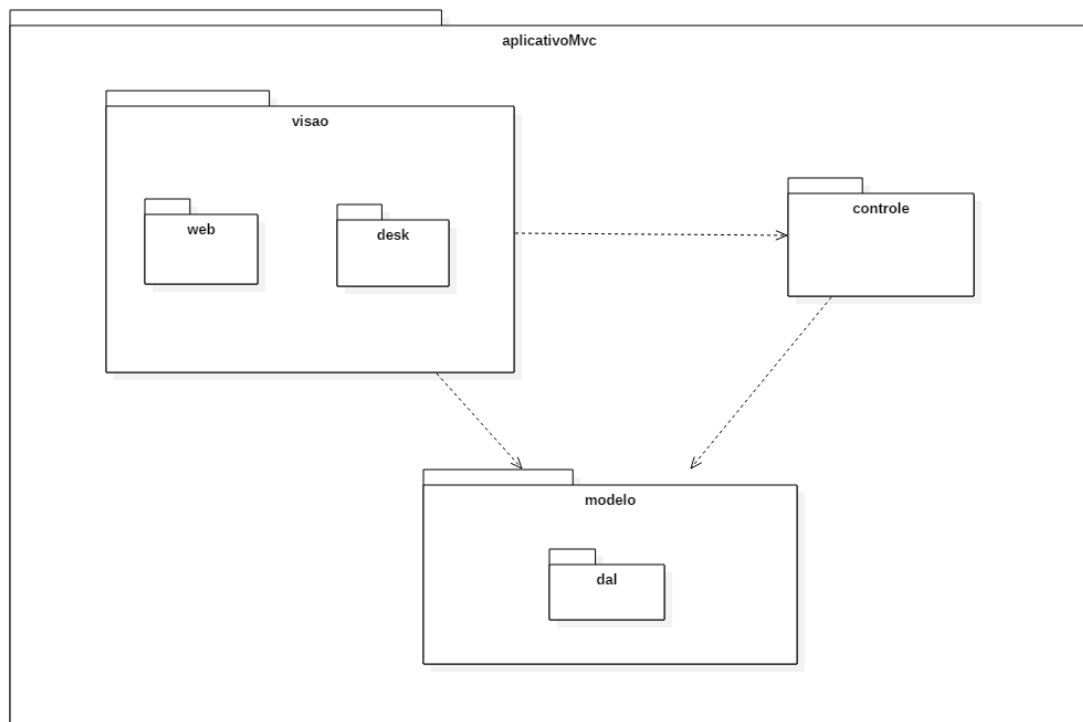


A arquitetura MVC especifica que um aplicativo com interfaces gráficas com o usuário deve ser organizado nos pacotes modelo, visão e controle.

- **Modelo:** Neste pacote definimos as classes, interfaces e subpacotes relacionados ao modelo (abstração) do problema do cliente. Neste pacote costumam aparecer as entidades (classes que armazenam e transportam dados, às vezes chamadas de DTOs – Data Transfer Objects) assim como classes que permitem a persistência de dados no banco de dados (às vezes chamadas DAOs – Data Access Objects);
- **Visão:** Neste pacote criamos as classes de fronteira que implementam as interfaces gráficas com o usuário. Caso haja mais de uma modalidade de interação (desktop, web, móvel, realidade virtual, realidade aumentada, vocal, telepática, etc.) podemos criar subpacotes, um para cada modalidade;
- **Controle:** Neste pacote estão as classes, interfaces, subpacotes relacionados à organização do fluxo de informações e de troca de mensagens entre as várias partes do sistema. Por exemplo, são as classes de controle que sabem qual é o banco de dados que está configurado na implantação em execução atualmente.

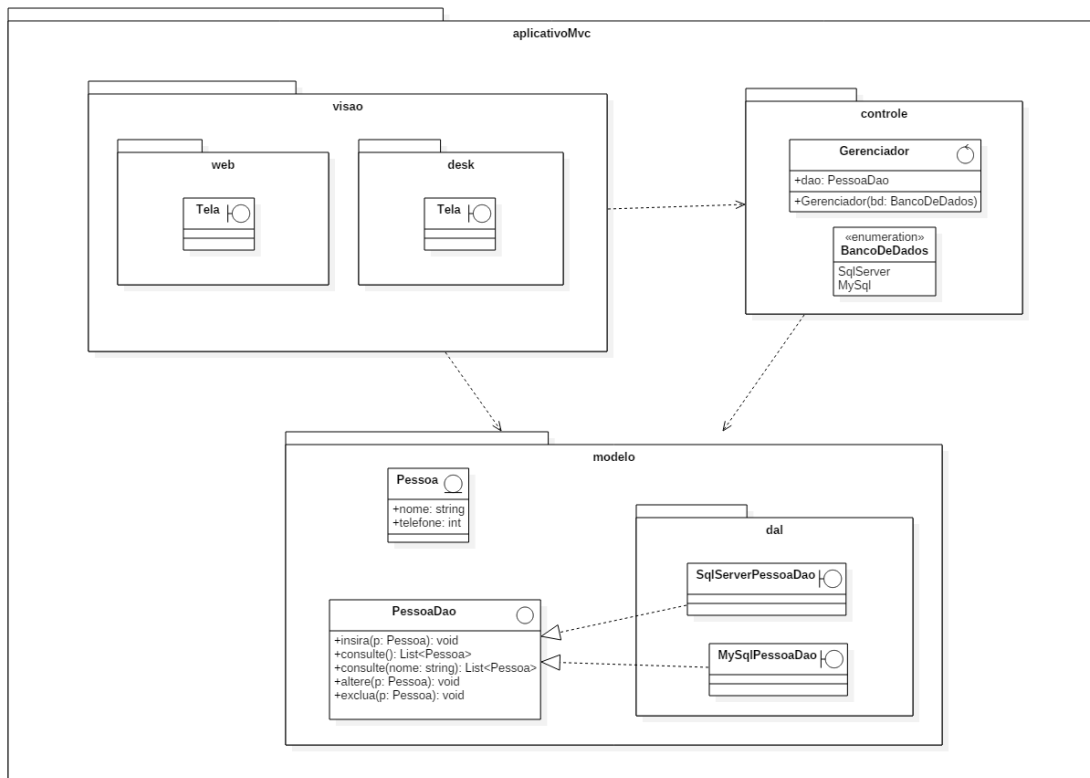
As setas de dependência indicam que a maioria das classes de um pacote dependem da maioria das classes do outro pacote. Por exemplo, observando a Figura 1, vemos que a maioria das classes do pacote de visão depende da maioria das classes dos pacotes de controle e modelo. Na prática, as dependências entre pacotes indicam que deve haver um “using” em cada classe que depende para os pacotes dependentes.

Figura 3: Diagrama de pacotes do aplicativo MVC.



No diagrama de pacotes acima desenvolvemos o modelo MVC para incorporar no pacote de visão os subpacotes responsáveis pelas interfaces gráficas com o usuário em web (ASP.Net) e em desktop (C#). No pacote modelo vamos criar o subpacote `dal` (Data Access Layer) para conter as classes de acesso a banco de dados.

Figura 4: Diagrama de classes do aplicativo MVC.



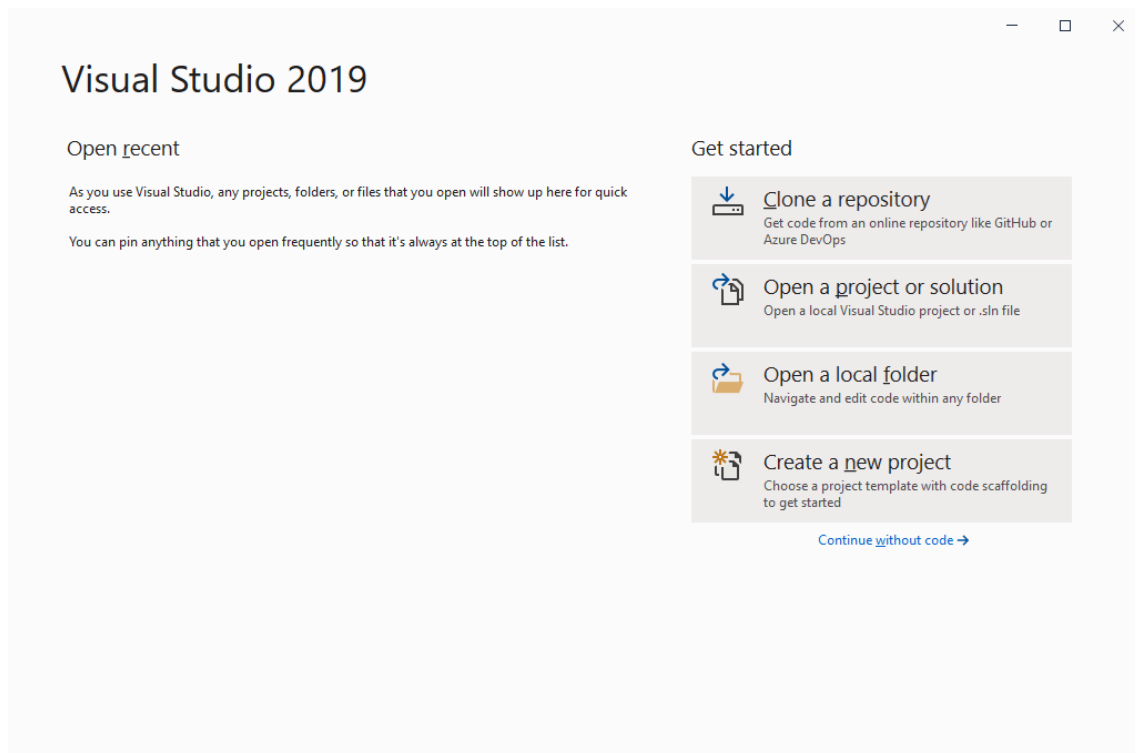
Finalmente, no diagrama de classes acima apresentamos as classes que serão desenvolvidas neste exemplo.

No pacote de visão, há duas classes **Tela**, uma para cada modalidade de interação.

No pacote modelo temos a entidade **Pessoa** e a interface **PessoaDao**, que expõe os serviços que uma classe Dao deve implementar. Já no subpacote **dal**, apresentamos duas classes DAO, uma para cada banco de dados. Ambas as classes DAO realizam a interface **PessoaDao**, ou seja, devem obrigatoriamente sobrescrever os métodos definidos na interface. Em nosso exemplo, vamos criar uma classe DAO de “mentirinha”, que vai apenas simular a manipulação do banco de dados, pois este assunto é abordado em outra disciplina.

No pacote controle, temos a classe **Gerenciador** que irá oferecer os serviços CRUD por meio de seu atributo **dao** que irá armazenar uma instância da classe DAO apropriada para o banco de dados implantado atualmente. Para tanto, um objeto da classe **Gerenciador** será instanciado definindo qual o banco de dados atual passando para o construtor o enumerador **BancoDeDados** apropriado.

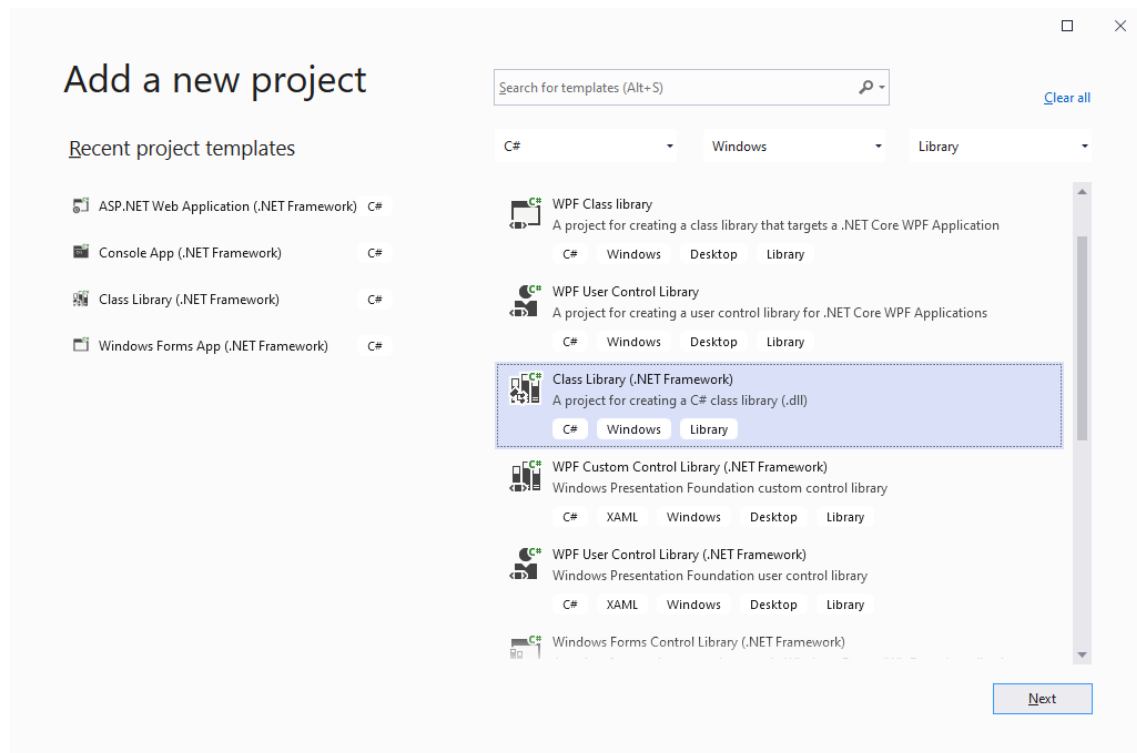
Figura 5: Página inicial do Visual Studio.



Vamos iniciar nossa solução clicando em [Create a new Project].

A Microsoft utiliza uma ferramenta de tradução automática para traduzir o Visual Studio para o português. Estas ferramentas ainda deixam muito a desejar para texto coloquial. Mas quando se trata de texto técnico, como os menus, botões e, principalmente, as documentações das linguagens, esta ferramenta torna o uso da ferramenta extremamente difícil. Recomendo que ferramentas profissionais, como o Visual Studio, sejam sempre usadas em seu idioma original.

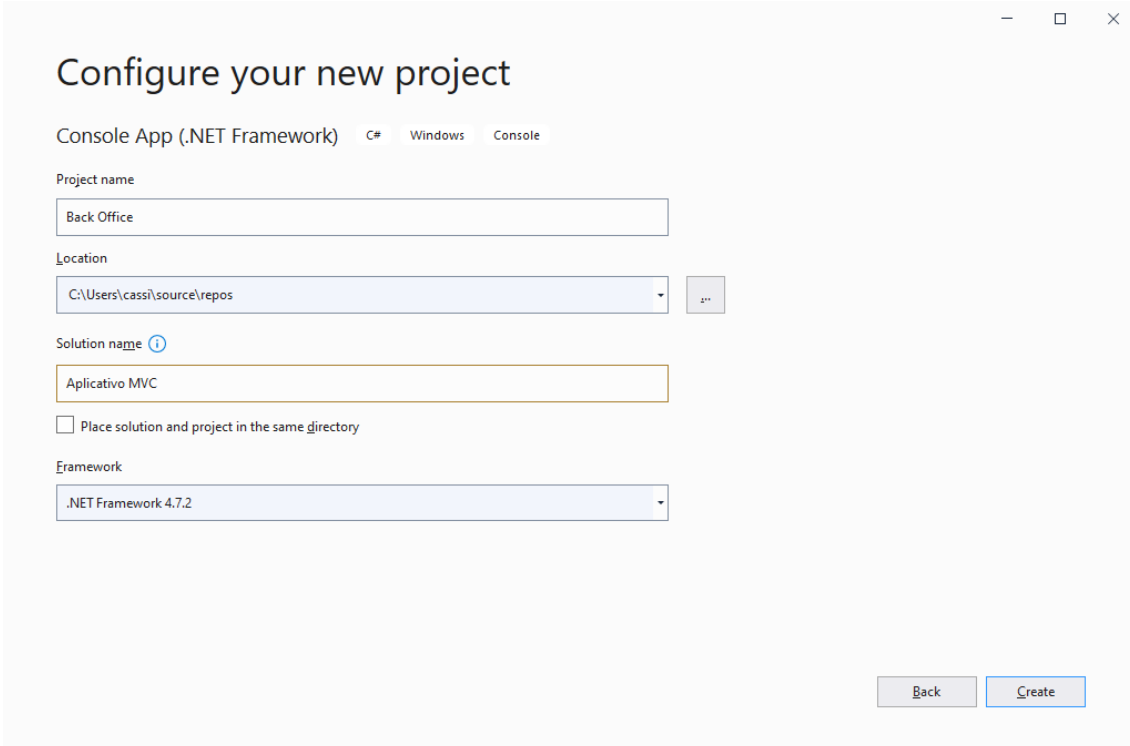
Figura 6: Criando um projeto.



A seguir, selecione a linguagem C#, a plataforma Windows e o tipo de projeto Library. Na lista de projetos com estes filtros aplicados, selecione Class Library (.NET Framework) para criar os pacotes com as classes que não estarão associadas a ferramentas de modelagem de interfaces gráficas.

Cuidado: Selecione o modelo que utiliza o .NET Framework. Não selecione o modelo que utiliza o .NET Core, que é o framework específico para aplicativos de servidor.

Figura 7: Configuração do projeto de back-office.



Configure your new project

Console App (.NET Framework) C# Windows Console

Project name

Back Office

Location

C:\Users\cassi\source\repos

Solution name ⓘ

Aplicativo MVC

☐ Place solution and project in the same directory

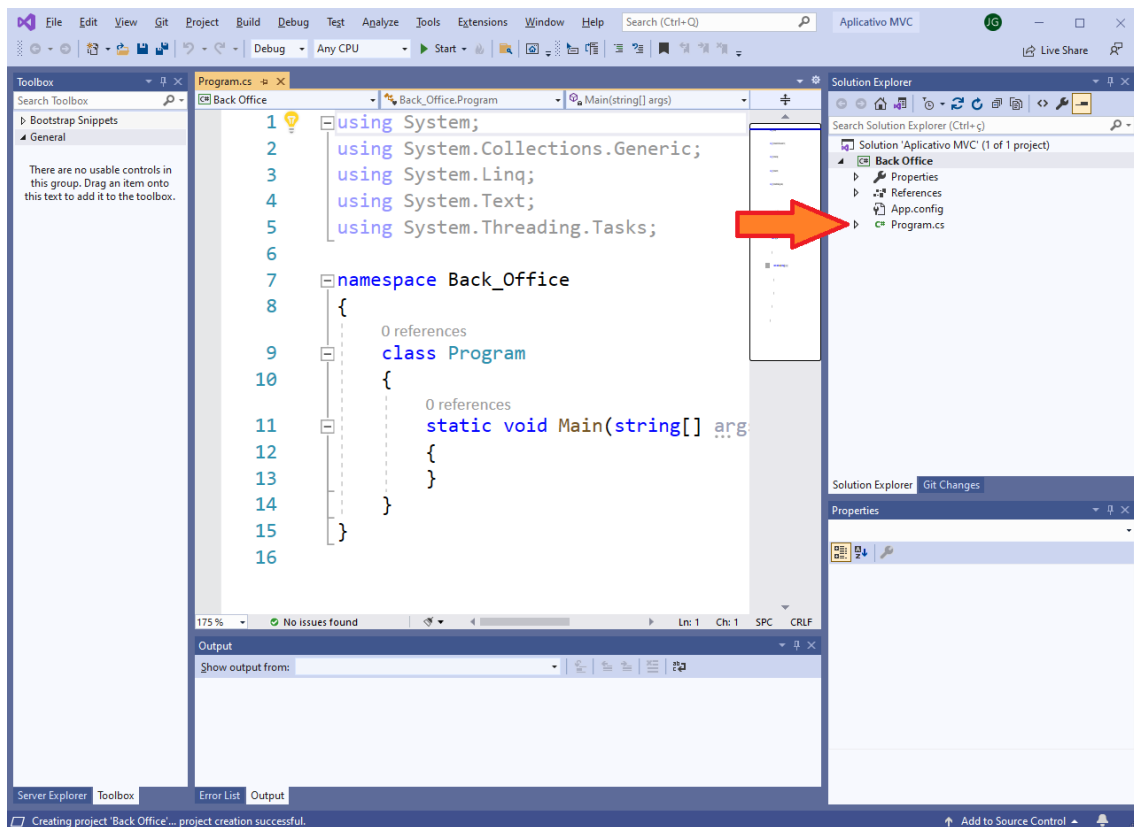
Framework

.NET Framework 4.7.2

Back Create

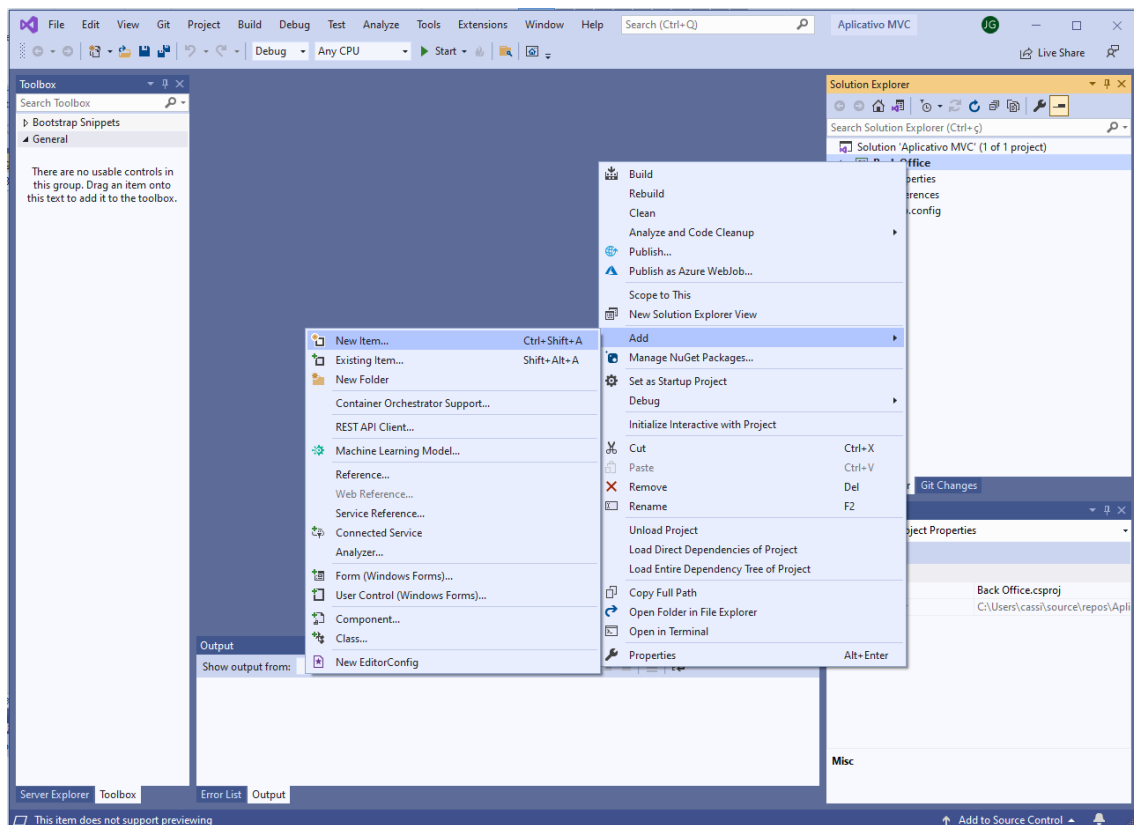
Nesta tela, defina o nome do seu projeto como “Back Office”. É neste projeto que vamos criar os pacotes controle e modelo. O pacote de visão será acrescentado mais tarde. Como nome da solução, use “Aplicativo MVC”. A solução é o nome do seu produto completo, por exemplo, Office. Já os projetos são as diferentes assemblies da sua solução, por exemplo, Word, Excel e Powerpoint.

Figura 8: Limpando o projeto.



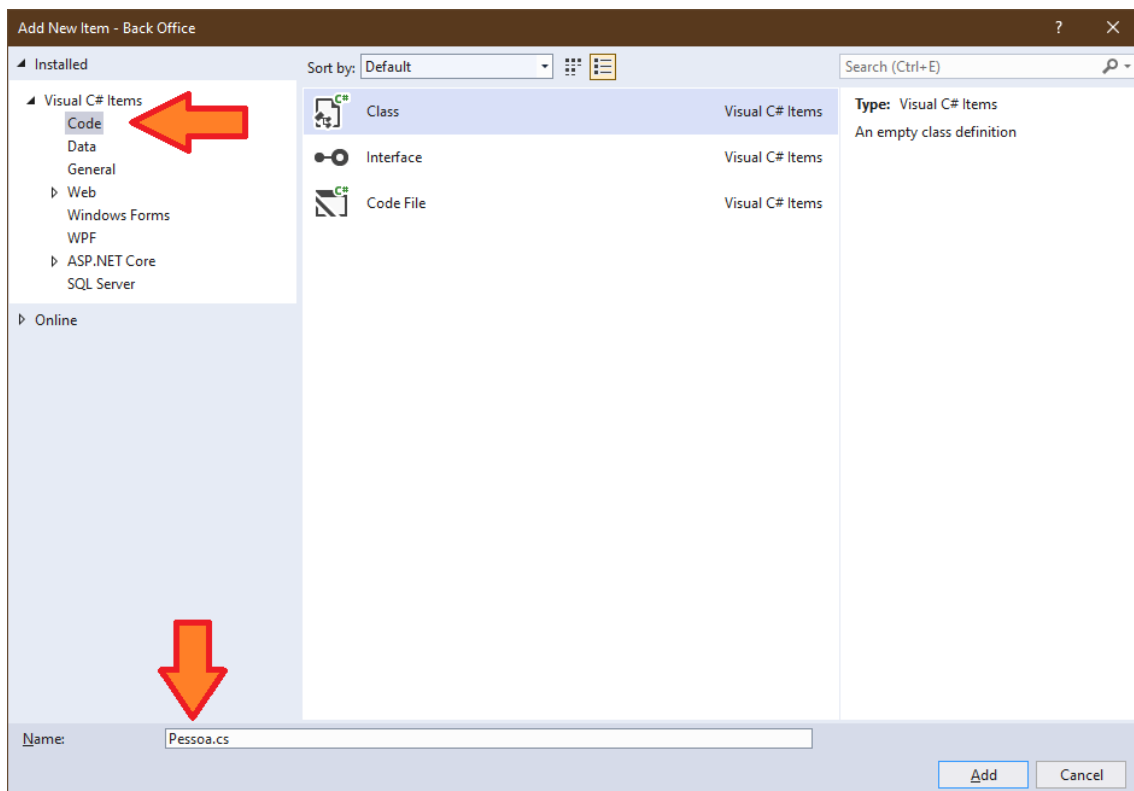
O Visual Studio cria a solução Aplicativo MVC com o projeto Back Office dentro dele. Também é criado o Program.cs. Clique sobre ele com o botão direito e apague este código. A seguir, vamos criar nossa classe Pessoa. Clique com o botão direito sobre o projeto Back Office e selecione [Add] -> [New Item...].

Figura 9: Adicionando um novo item ao projeto.



A seguir, filtre os modelos por Code, escolha Classe e dê o nome Pessoa.cs à sua classe.

Figura 10: Criando a classe Pessoa.



Na classe Pessoa.cs que você acabou de criar, escreva o código da classe Pessoa.

Figura 11: Código da classe Pessoa.

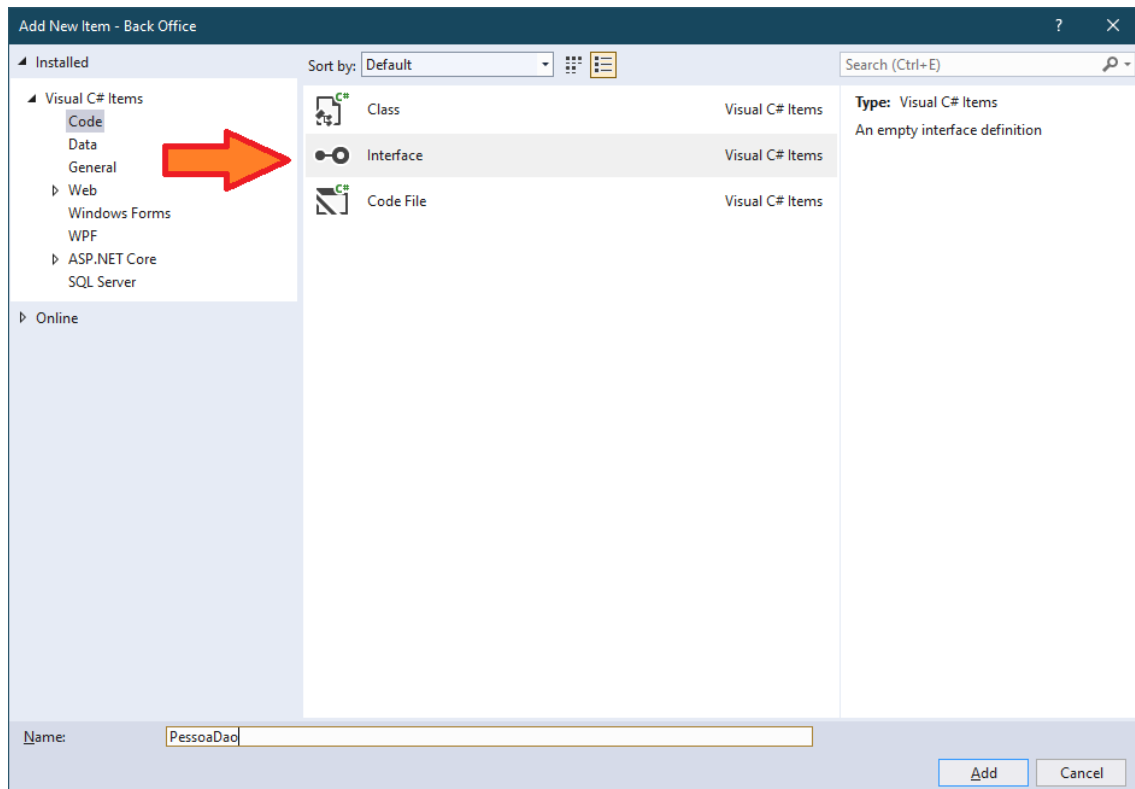
```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace aplicativoMVC.modelo
8  {
9      19 references
10     public class Pessoa
11     {
12         6 references
13         public string nome { get; set; }
14         5 references
15         public int telefone { get; set; }
16         0 references
17         public override bool Equals(object obj)
18         {
19             bool saida = false;
20
21             if (typeof(Pessoa) == obj.GetType()){
22                 Pessoa pessoa = (Pessoa)obj;
23                 if (this.nome.Equals(pessoa.nome) && this.telefone == pessoa.telefone)
24                 {
25                     saida = true;
26                 }
27             }
28
29             return saida;
30         }
31         0 references
32         public override string ToString()
33         {
34             return nome + " - " + telefone;
35         }
36     }
37 }
```

Note que namespace define o nome do pacote. Confira a Figura 4. Note também que as inclusões (using) estão em cinza porque não foram usadas. Você pode apaga-las se quiser. O Visual Studio indica quantas vezes a classe e os atributos foram referenciados (0 references). Este texto foi escrito automaticamente pelo Visual Studio e não faz parte do código (não o copie!). Finalmente, estamos usando as propriedades sem qualquer lógica ({get; set;}). A prática correta é usar as propriedades para proteger o conteúdo dos atributos evitando o armazenamento de valores inválidos.

Na linha 13 estamos sobrescrevendo o método Equals() herdado da classe Object. Este método retorna true se o parâmetro obj for da classe Pessoa e seus atributos nome e telefone forem iguais a nome e telefone da instância atual.

Vamos agora criar a interface PessoaDao.

Figura 12: Criando a interface PessoaDao.



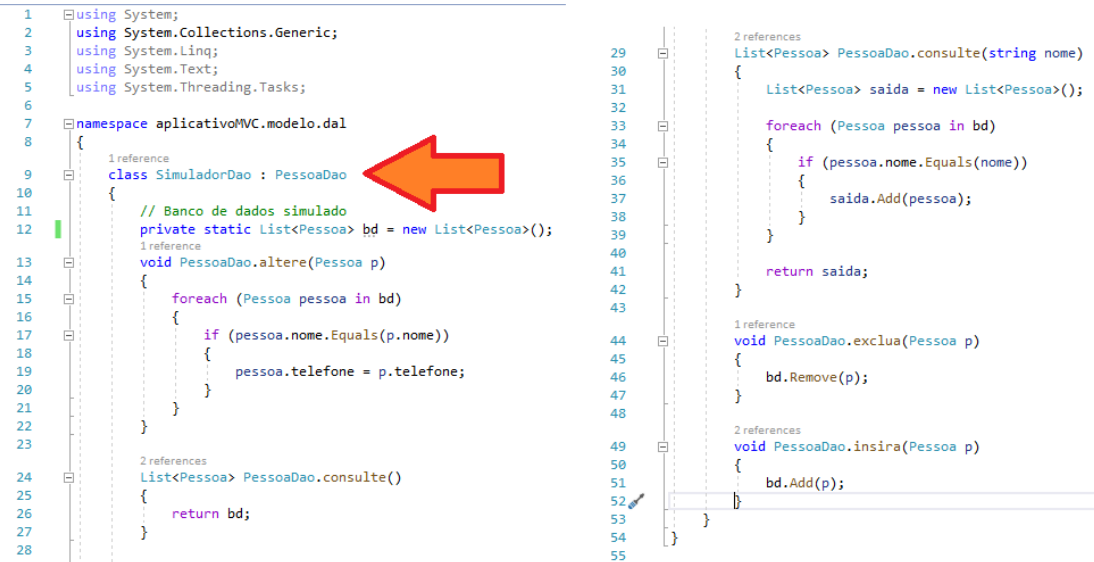
Escreva o código abaixo na interface PessoaDao;

Figura 13: Código da interface PessoaDao.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.Collections;
7
8  namespace aplicativoMVC.modelo
9  {
10     0 references
11     public interface PessoaDao
12     {
13         0 references
14         void insira(Pessoa p);
15         0 references
16         List<Pessoa> consulte();
17         0 references
18         List<Pessoa> consulte(string nome);
19         0 references
20         void altere(Pessoa p);
21         0 references
22         void exclua(Pessoa p);
23     }
24 }
```

A classe List pertence ao pacote System.Collections.Generic, não se esqueça de fazer sua importação (linha 2). Esta classe funciona como se fosse um vetor (array) que pode mudar de tamanho conforme elementos são adicionados ou removidos ela. Usando o argumento de tipo <Pessoa> um objeto da classe List <Pessoa> só irá armazenar objetos da classe Pessoa e devolverá objetos da mesma classe Pessoa.

Figura 14: Código da classe SimuladorDao.



```

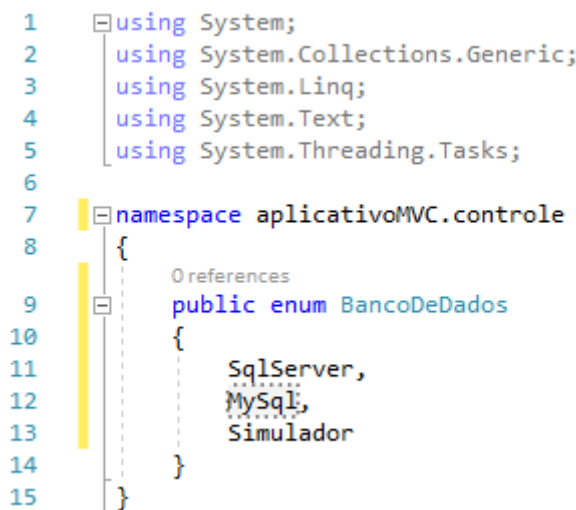
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace aplicativoMVC.modelo.dal
8  {
9      1 reference
10     class SimuladorDao : PessoaDao
11     {
12         // Banco de dados simulado
13         private static List<Pessoa> bd = new List<Pessoa>();
14
15         1 reference
16         void PessoaDao.altere(Pessoa p)
17         {
18             foreach (Pessoa pessoa in bd)
19             {
20                 if (pessoa.nome.Equals(p.nome))
21                 {
22                     pessoa.telefone = p.telefone;
23                 }
24             }
25
26             2 references
27             List<Pessoa> PessoaDao.consulte()
28             {
29                 return bd;
30             }
31
32             2 references
33             List<Pessoa> PessoaDao.consulte(string nome)
34             {
35                 List<Pessoa> saida = new List<Pessoa>();
36
37                 foreach (Pessoa pessoa in bd)
38                 {
39                     if (pessoa.nome.Equals(nome))
40                     {
41                         saida.Add(pessoa);
42                     }
43                 }
44
45                 return saida;
46             }
47
48             1 reference
49             void PessoaDao.exclua(Pessoa p)
50             {
51                 bd.Remove(p);
52             }
53
54             2 references
55             void PessoaDao.insira(Pessoa p)
56             {
57                 bd.Add(p);
58             }
59         }
60     }
61 }

```

Escreva o código da classe SimuladorDao, que irá simular uma operação de persistência em banco de dados. Não se esqueça de definir que a classe está no pacote aplicativoMVC.modelo.dal e que ela é uma realização da interface PessoaDao.

Crie o enumerador BancoDeDados. Para tanto, crie uma nova classe e altere o código da classe conforme o código abaixo.

Figura 15: Código do enumerador BancoDeDados.



```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace aplicativoMVC.controle
8  {
9      0 references
10     public enum BancoDeDados
11     {
12         SqlServer,
13         MySql,
14         Simulador
15     }
16 }

```

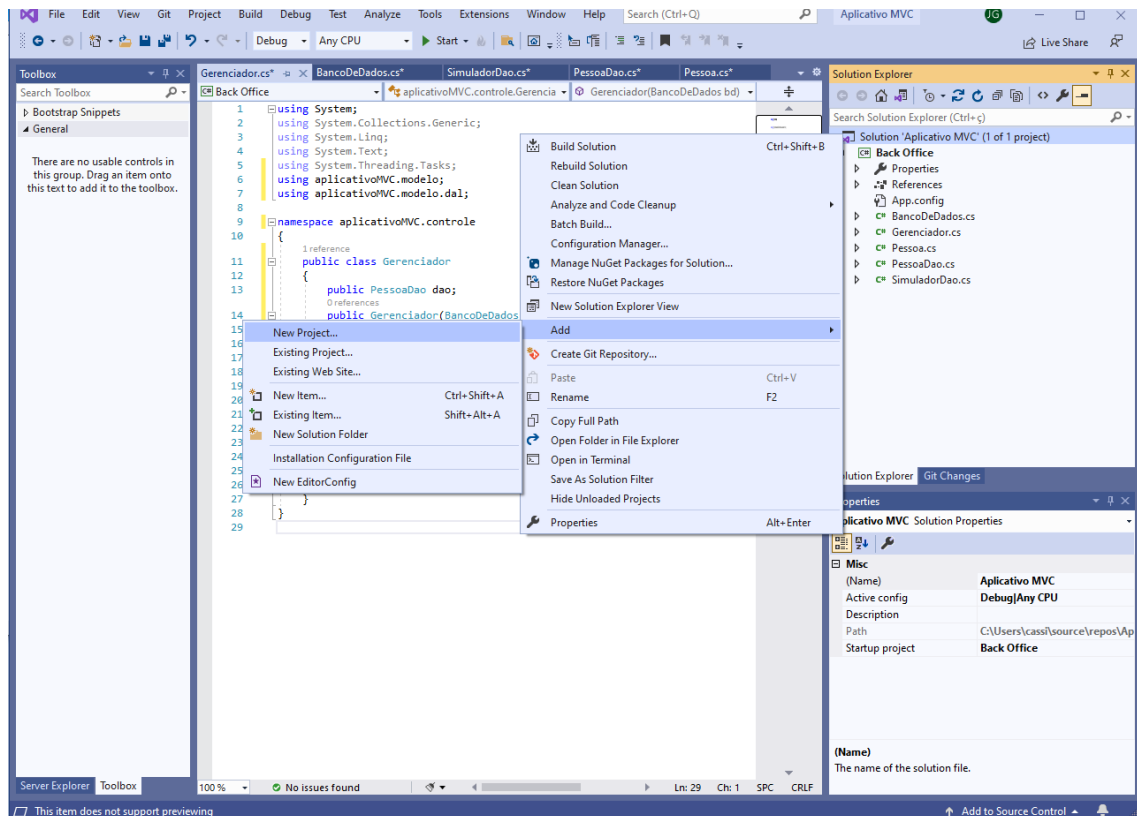
Agora crie a classe Gerenciador com o código apresentado a seguir.

Figura 16: Código da classe Gerenciador.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using aplicativoMVC.modelo;
7 using aplicativoMVC.modelo.dal;
8
9 namespace aplicativoMVC.controle
10 {
11     1 reference
12     public class Gerenciador
13     {
14         public PessoaDao dao;
15         0 references
16         public Gerenciador(BancoDeDados bd)
17         {
18             switch (bd)
19             {
20                 case BancoDeDados.Simulador:
21                     dao = new SimuladorDao();
22                     break;
23                 case BancoDeDados.MySql:
24                     throw new NotImplementedException("DAO para MySql não implementado");
25                 case BancoDeDados.SqlServer:
26                     throw new NotImplementedException("DAO para SQL Server não implementado");
27             }
28         }
29     }
30 }
```

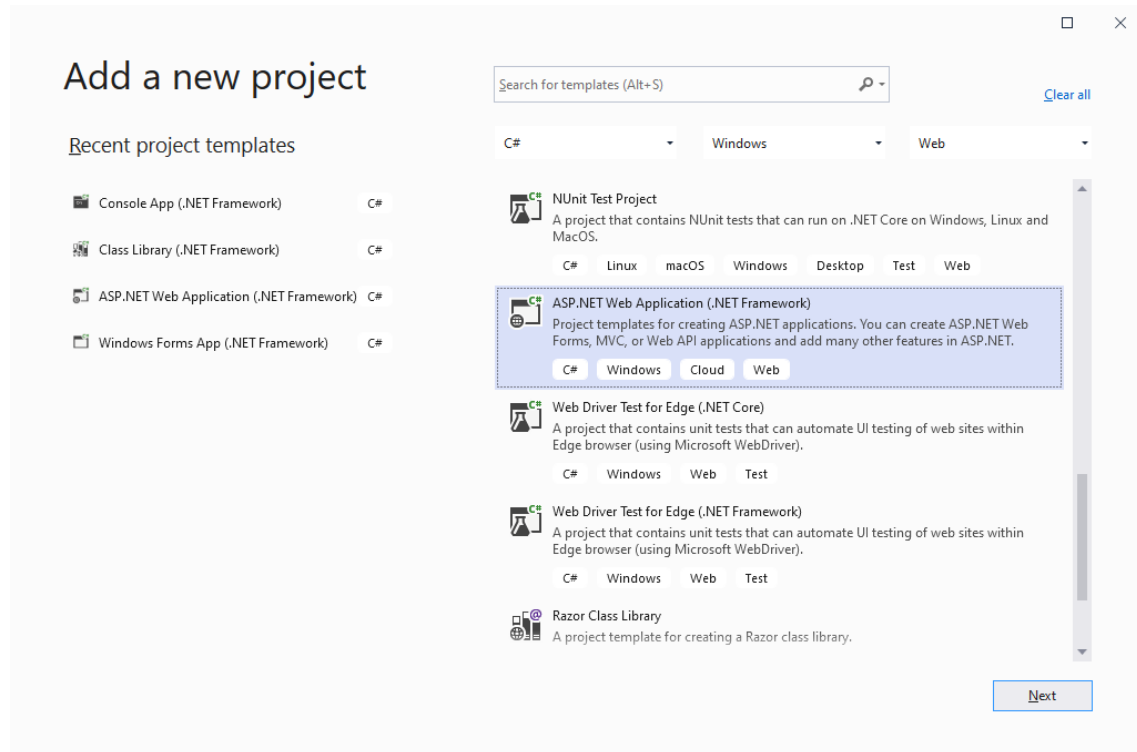
Neste ponto terminamos o back-office de nosso aplicativo. Agora vamos criar a interface gráfica com o usuário usando ASP.Net. Para tanto, clique com o botão direito sobre a solução Aplicativo MVC e selecione [Add] -> [New Project...]

Figura 17: Inserindo um novo projeto.



A seguir, mude o filtro para apresentar projetos C#, Windows e Web. Selecione o modelo ASP.NET Web Application (.NET Framework). Novamente, cuidado para não selecionar um modelo que use o .NET Core.

Figura 18: Selecionando o modelo ASP.NET Web Application (.NET Framework).



Na próxima janela, chame seu projeto de Tela web. Note que desta vez não é preciso definir o nome da solução (ela já existe).

Figura 19: Nomeando o projeto ASP.NET.

Configure your new project

ASP.NET Web Application (.NET Framework) C# Windows Cloud Web

Project name

Tela web

Location

C:\Users\cassi\source\repos\Aplicativo MVC

Framework

.NET Framework 4.7.2

Back Create

Na janela seguinte, selecione o modelo Empty (vazio). Note que há outros modelos, inclusive um MVC. Neste caso, o modelo MVC irá criar um aplicativo padronizado quase completo, mas que não seria apropriado para o nosso problema particular. Em nosso exemplo estamos criando nossa própria arquitetura baseada no modelo MVC.

Figura 20: Selecionando um modelo de aplicativo web.

Create a new ASP.NET Web Application

Empty
An empty project template for creating ASP.NET applications. This template does not have any content in it.

Web Forms
A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

MVC
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Web API
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Single Page Application
A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

Authentication
No Authentication
[Change](#)

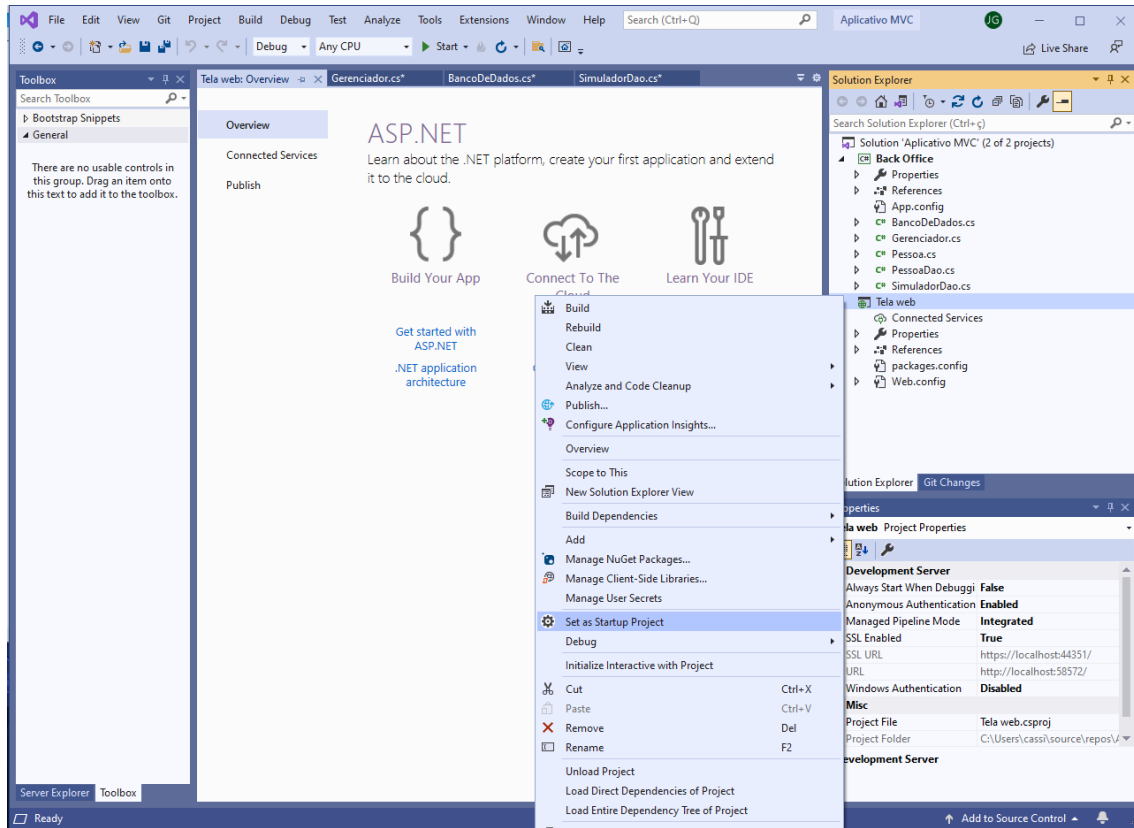
Add folders & core references
☐ Web Forms
☐ MVC
☐ Web API

Advanced
☒ [Configure for HTTPS](#)
☐ Docker support
(Requires [Docker Desktop](#))
☐ Also create a project for [unit tests](#)
☒ [Tela web.Tests](#)

[Back](#) [Create](#)

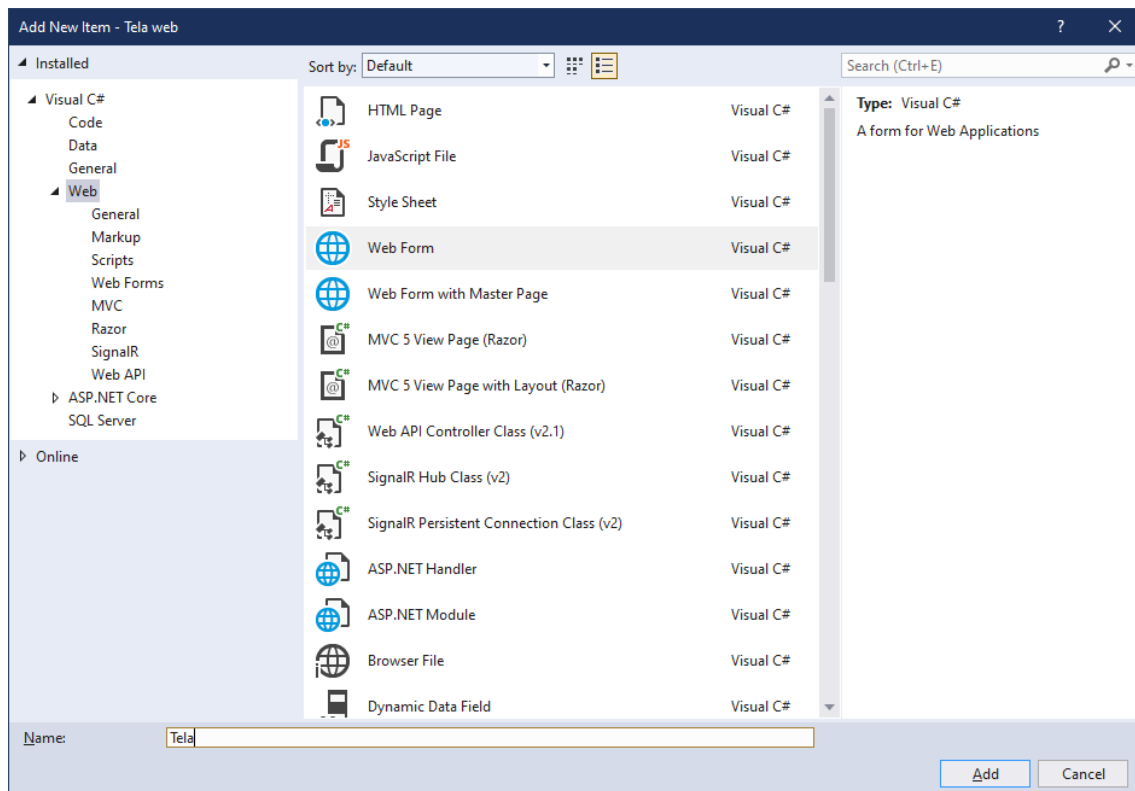
Agora a nossa solução apresenta dois projetos, o Back Office e o Tela web. Clique no projeto Tela web com o botão direito e selecione [Set as Startup Project], para que o projeto web seja executado quando estivermos prontos para testar nosso aplicativo.

Figura 21: Definindo o projeto web como projeto de início.



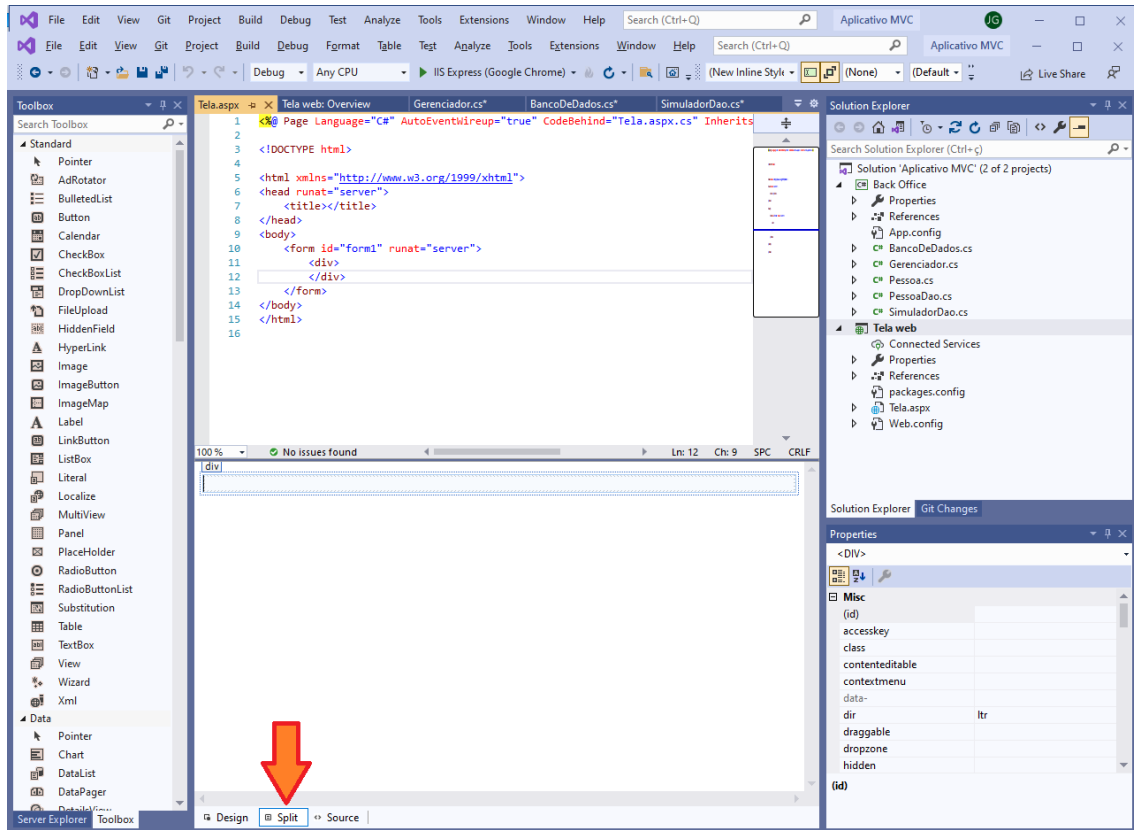
Clique novamente com o botão direito sobre o projeto Tela web, selecione [Add] -> [New item...]. Escolha um Web Form e defina seu nome como Tela.

Figura 22: Inserindo a Tela web.



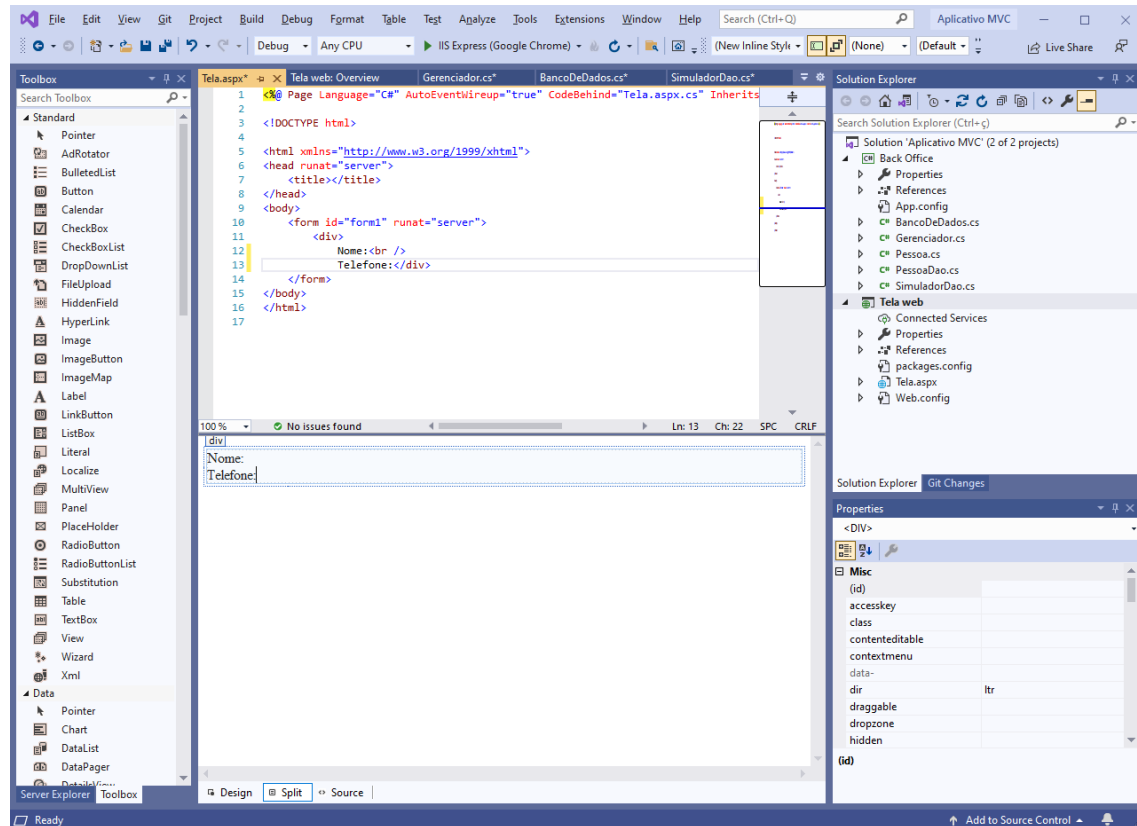
Você pode visualizar sua página ASP.NET por código, pelo editor ou por ambos.

Figura 23: Seleção da visualização de código fonte, de editor ou de ambos.



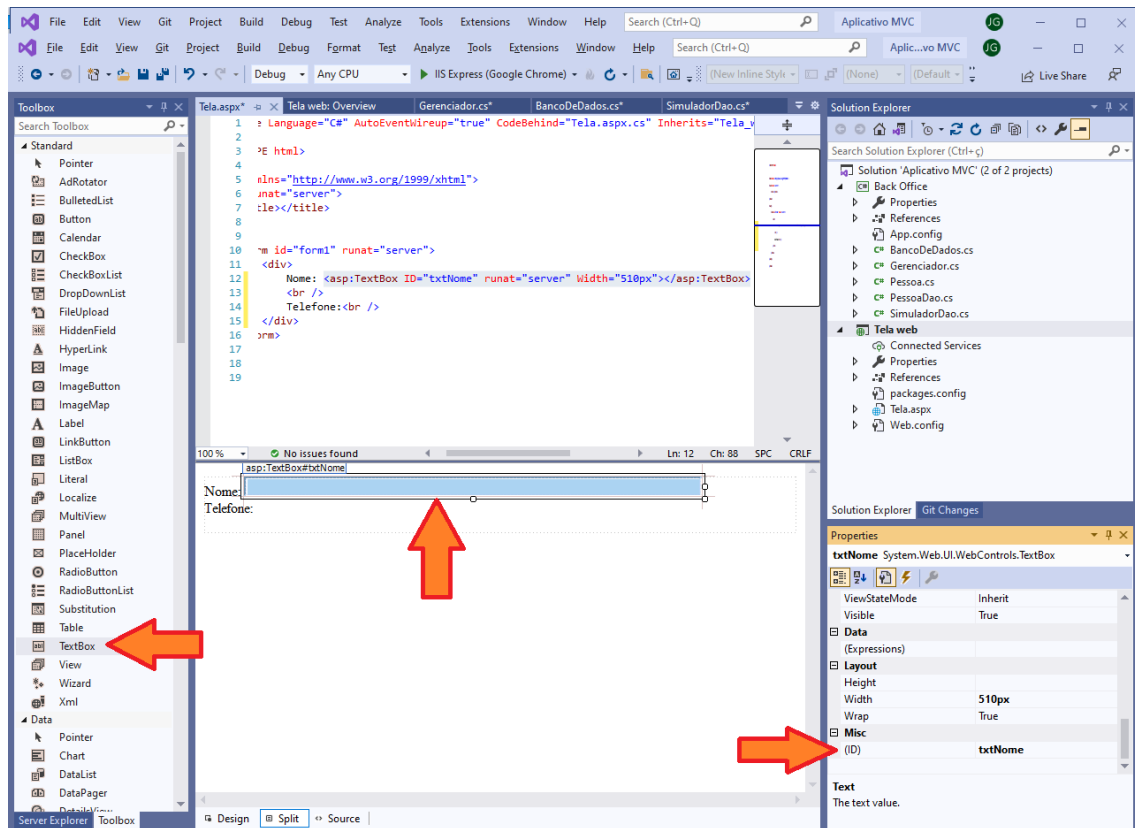
No elemento div do editor visual, digite “Nome:[ENTER]Telefone:”. Acompanhe a mudança no editor de código.

Figura 24: Inserindo texto na tela de usuário.



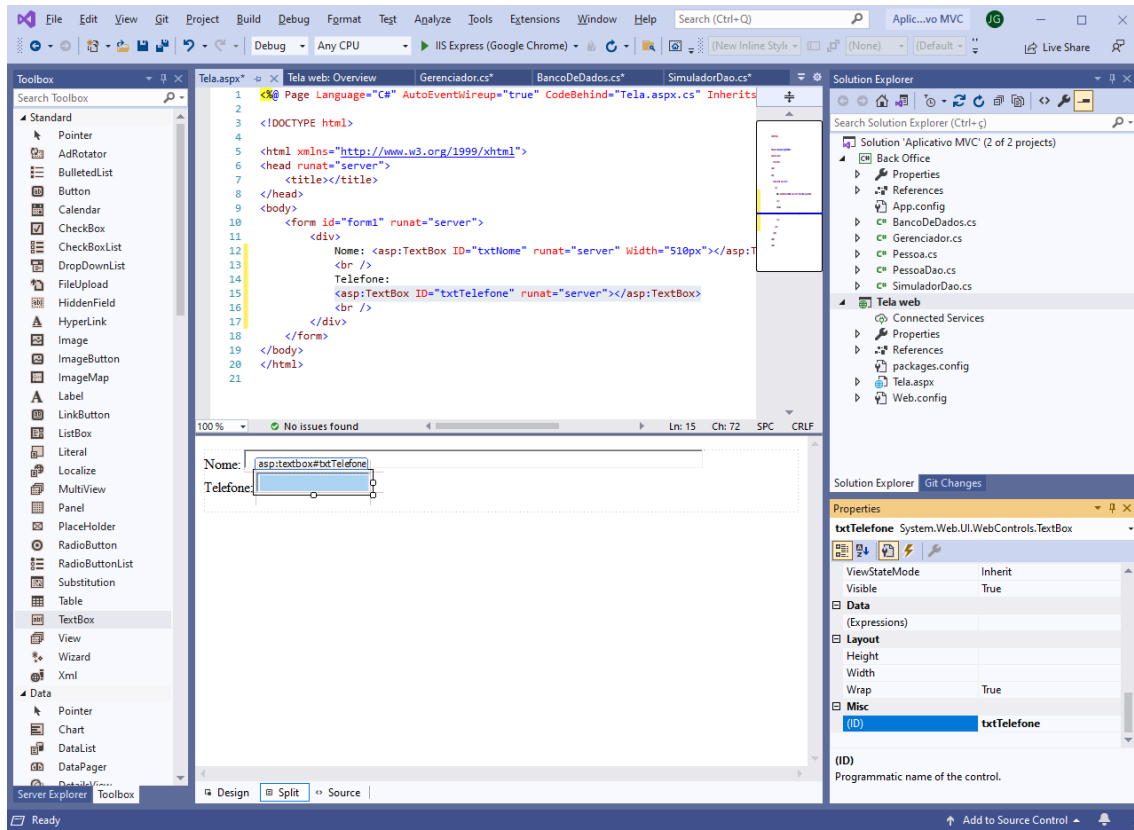
Arraste um TextBox para o lado do texto “Nome:”. Altere o identificador (ID) deste TextBox para txtNome. É com este nome que vamos referenciar o campo no código. Você pode dimensionar o tamanho do TextBox como quiser.

Figura 25: Inserindo o TextBox txtNome.



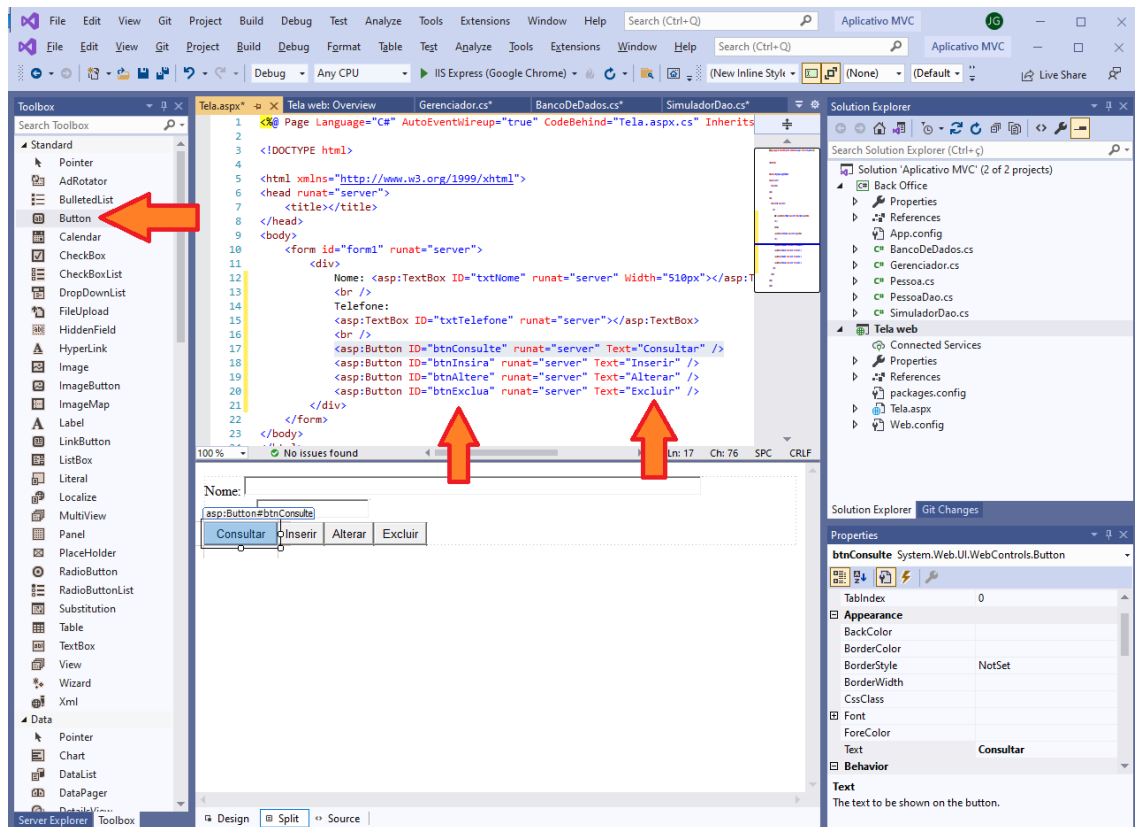
Repita o processo para inserir o txtTelefone.

Figura 26: Inserindo o TextBox txtTelefone.



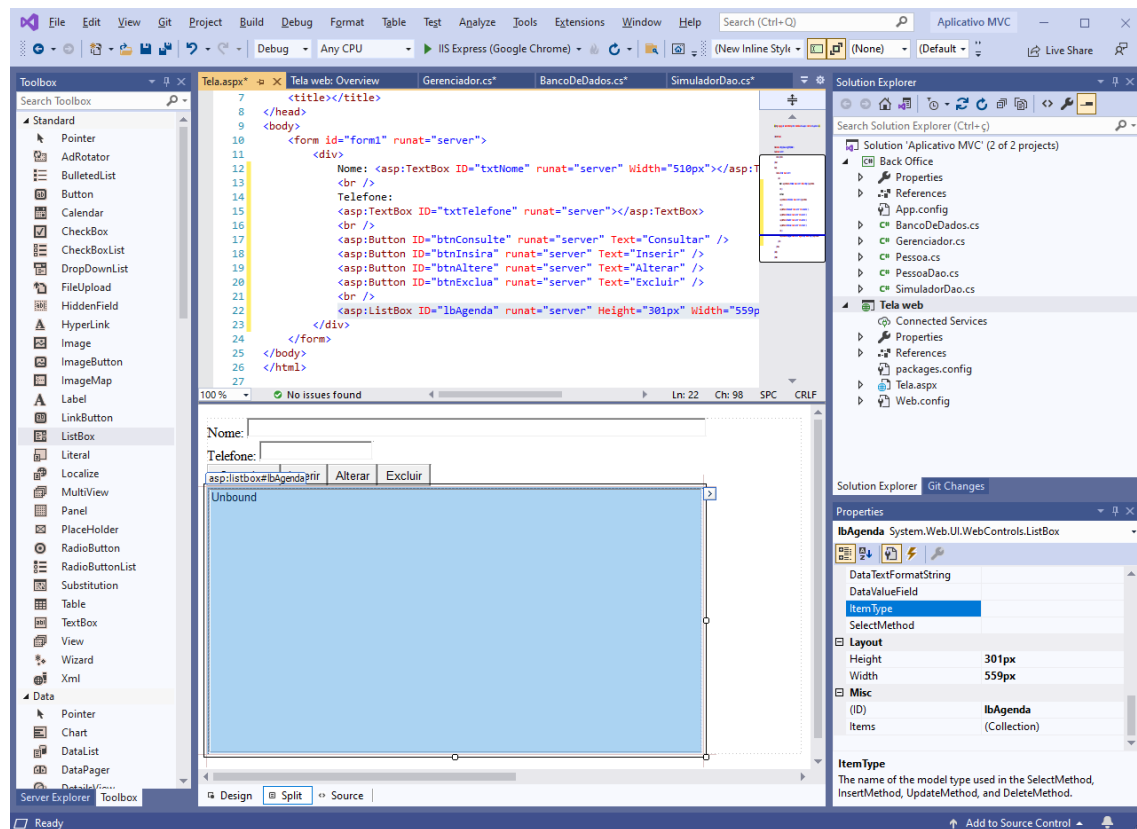
Abaixo da linha “Telefone:”, insira quatro botões, definindo seus atributos Text para Consultar, Inserir, Alterar e Excluir respectivamente. Altere também seus IDs para btnConsulte, btnInsira, btnAltere e btnExclua respectivamente. Você pode editar os atributos Text e ID tanto pelo painel de Propriedades quanto pelo editor de código.

Figura 27: Inserindo os botões das operações CRUD.



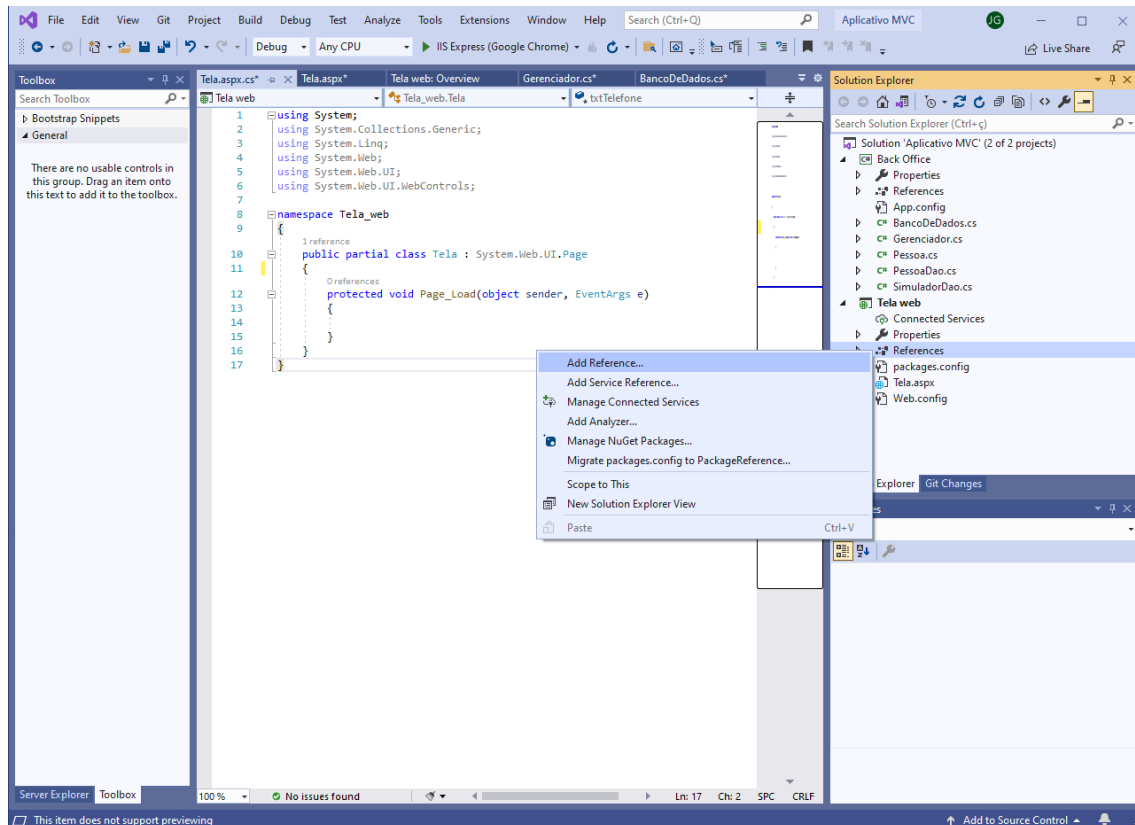
Adicione um ListBox abaixo dos botões e defina seu ID para lbAgenda. Altere seu tamanho como quiser.

Figura 28: Inserindo o ListBox lbAgenda.



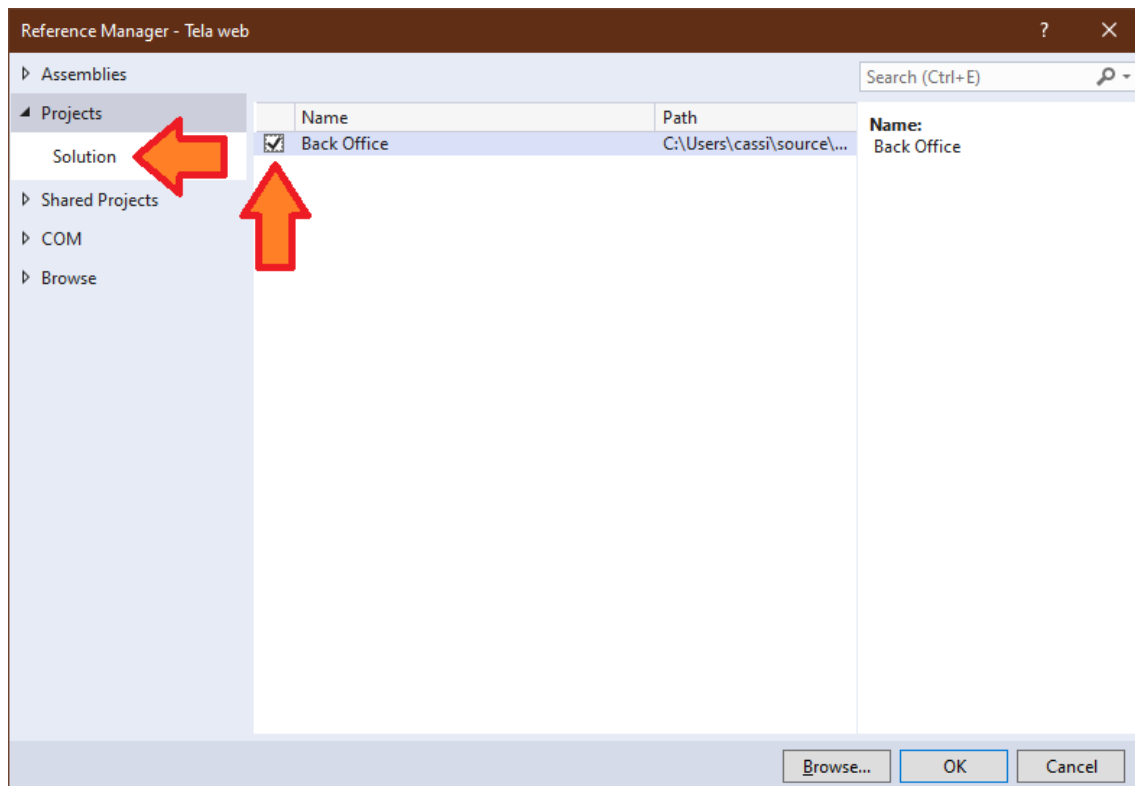
Agora, dê um clique duplo em uma área em branco do editor. O Visual Studio irá abrir o código `Tela.aspx.cs` que é uma classe C# associada ao layout `Tela.aspx`. Ele também cria o método `Page_Load()` que é executado quando a página é carregada. Vamos iniciar um objeto da classe `Gerenciador` neste método. Para usar a classe `Gerenciador`, que está definida em outro projeto, precisamos inicialmente incorporar tal projeto no projeto atual. Para tanto, clique com o botão direito sobre `References` do projeto `Tela Web`.

Figura 29: Inserindo uma referência ao projeto.



A seguir, selecione um projeto de sua solução e, na lista apresentada, selecione o projeto que deseja incorporar. No momento, o único projeto disponível é Back Office.

Figura 30: Inserindo uma referência ao projeto Back Office.



Agora podemos fazer a importação dos pacotes daquele projeto e completar o código da seguinte maneira.

Figura 31: Criando uma instância do Gerenciador na Tela Web.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.UI;
6  using System.Web.UI.WebControls;
7  using aplicativoMVC.controle;
8
9  namespace Tela_web
10 {
11     1 reference
12     public partial class Tela : System.Web.UI.Page
13     {
14         Gerenciador gerenciador;
15         0 references
16         protected void Page_Load(object sender, EventArgs e)
17         {
18             gerenciador = new Gerenciador(BancoDeDados.Simulador);
19         }
20     }
21 }
```

Volte ao editor da página e dê um clique duplo em cada um dos quatro botões de modo a criar os métodos que serão executados ao se clicar em cada um. A seguir, complete o código da classe conforme o exposto abaixo.

Figura 32: Código da classe Tela.aspx.cs.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Web.UI;
6 using System.Web.UI.WebControls;
7 using aplicativoMVC.controle;
8 using aplicativoMVC.modelo;
9
10 namespace Tela_web
11 {
12     2 references
13     public partial class Tela : System.Web.UI.Page
14     {
15         Gerenciador gerenciador;
16         0 references
17         protected void Page_Load(object sender, EventArgs e)
18         {
19             gerenciador = new Gerenciador(BancoDeDados.Simulador);
20
21         0 references
22         protected void btnConsulte_Click(object sender, EventArgs e)
23         {
24             List<Pessoa> pessoas;
25             if (txtNome.Text.Equals(""))
26             {
27                 pessoas = gerenciador.dao.consulte();
28             }
29             else
30             {
31                 pessoas = gerenciador.dao.consulte(txtNome.Text);
32             }
33             lbAgenda.Items.Clear();
34             foreach (Pessoa p in pessoas)
35             {
36                 lbAgenda.Items.Add(p.ToString());
37             }
38
39         0 references
40         protected void btnInsira_Click(object sender, EventArgs e)
41         {
42             Pessoa p = new Pessoa();
43             p.nome = txtNome.Text;
44             p.telefone = Convert.ToInt32(txtTelefone.Text);
45             gerenciador.dao.insira(p);
46             txtNome.Text = "";
47             txtTelefone.Text = "";
48
49         0 references
50         protected void btnAltere_Click(object sender, EventArgs e)
51         {
52             Pessoa p = new Pessoa();
53             p.nome = txtNome.Text;
54             p.telefone = Convert.ToInt32(txtTelefone.Text);
55             gerenciador.dao.altere(p);
56
57         0 references
58         protected void btnExclua_Click(object sender, EventArgs e)
59         {
60             Pessoa p = new Pessoa();
61             p.nome = txtNome.Text;
62             p.telefone = Convert.ToInt32(txtTelefone.Text);
63             gerenciador.dao.exclua(p);
64         }
65     }
66 }
```

Execute seu aplicativo clicando no botão Start Debugging e veja o resultado.

Figura 33: Executando o aplicativo.

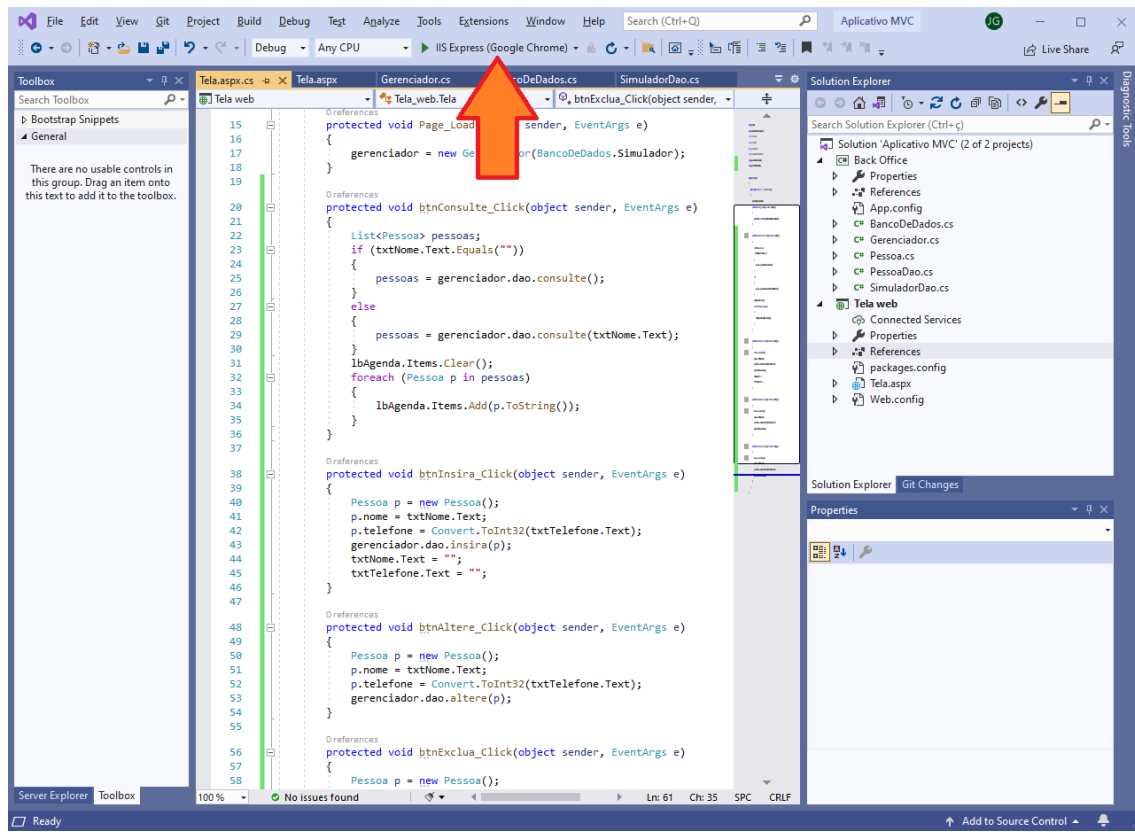


Figura 34: Aplicativo em execução.

