

Apostila de Linguagem de Programação I

ASP.NET Core MVC 3.1 + SQL SERVER

Professor Eduardo Rosalém Marcelino

(2017 / 2022)

eduardormbr@gmail.com

pro6761@cefsa.edu.br

Conteúdo

Idioma	4
Principais teclas de atalho	4
Instalação do SQL SERVER – Configurando modo de autenticação Misto	5
1. Arquitetura de uma aplicação WEB.....	6
2. HTML.....	6
3. CSS.....	6
4. Javascript.....	6
5. Criando o primeiro projeto Asp.Net Core – Projeto soma	7
6. Criando um projeto para o cálculo de tabuada	9
Exercício	9
7. Arquitetura Model View Controller	10
8. Asp.net MVC Core.....	10
Introdução ao ASP.NET Core	10
Por que escolher o ASP.NET Core?	11
9. Acesso ao banco de dados com C#.....	12
Projeto Windows Forms	13
Classes para transferência de dados entre as camadas da aplicação: ViewModel ou VO (Value Object) ou DTO (Data Transfer Object)	13
Classe de Conexão com o Banco de dados	13
Classe DAO para realização das operações de acesso ao banco de dados	15
Formulário de Cadastro	16
Exercícios:	17
10. SQL Injection	18
11. Evitando SQL Injection - Ajustes no cadastro de alunos.....	19

Classe DAO com parâmetros. Métodos excluir e alterar registros	20
Alterações no formulário para excluir e alterar	21
Exercícios:	21
12. Consultas.....	22
Redução de código.....	24
Exercícios:	24
13. Criando um projeto Asp.NET Core no Visual studio 2019	25
Model: Adicionando arquivos para acesso ao banco de dados	27
Controller	29
View (arquivos Razor)	30
Configurando o controller inicial	32
Executando o site.....	32
Exercícios:	32
14. Cadastrando um novo aluno.....	33
Exercícios:	35
15. Armazenando e recuperando valores nulo (null)	35
16. Exibindo erros	36
Exercícios:	37
17. Enviando parâmetros pela URL para o controller: QueryString.....	38
18. Consultando e alterando os dados do aluno	39
Exercícios:	40
19. Excluindo registros.....	41
20. Sugerindo o próximo código de aluno disponível	41
Exercícios:	42
21. ViewBag – Passando dados para a View.....	42
22. Desabilitando o campo Id nas alterações	43
Exercícios:	44
23. RAZOR e ASP.NET TAG HELPERS.....	44
O que são Auxiliares de Marca (TAG HELPERS)	44
24. Validando os campos do formulário.....	45
Exercícios:	47
25. Alterações no CRUD para realizar as operações utilizando Stored Procedures.....	48
Exercícios	50
26. Listando valores de uma tabela em uma caixa Combo	51
Exercícios	53
27. Herança aplicada aos Models e DAOs	54

Classe Ancestral para os todas as Models	54
Stored Procedures Genéricas	54
Stored procedures exclusivas para o cadastro de alunos	55
Classe DAO Padrão.....	55
Alterações no cadastro de alunos para se adequar ao novo padrão.	57
Exercícios	58
28. Controller Padrão.....	58
Aplicando o controlador Padrão em um CRUD de cidade.....	60
Ajustando o menu principal e criando um link para nossos CRUDs.....	62
Exercícios	64
29. Session	64
30. Controle de acesso.....	65
31. Salvando e Recuperando Imagens no Banco de Dados.....	69
32. Formato JSON	74
33. Carrinho de compras.....	75
34. Desenvolvimento de um cadastro Mestre Detalhe.....	82
35. AJAX	86
Aplicando AJAX ao projeto de soma de valores	86
Consumindo uma API via AJAX (API de consulta de CEP).....	88
Construindo uma consulta avançada de dados com AJAX	90
36. Consumindo APIs	96

Referências

[1] MACOTTE, C. An Atypical ASP.NET Core 5 Design Patterns Guide. Packt, 2020.

[2] MSDN



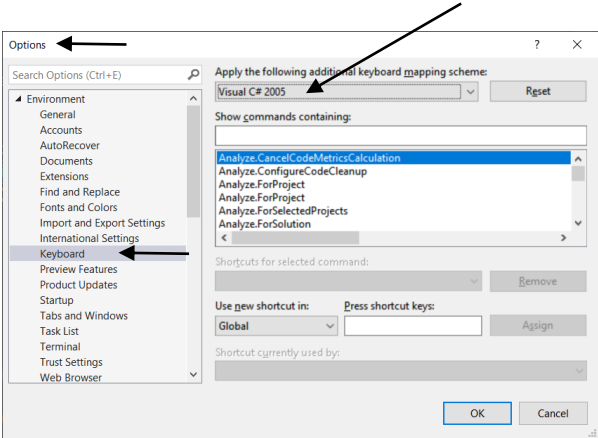
O idioma configurado no Visual Studio utilizado nesta apostila é o Inglês.

Principais teclas de atalho

Teclas de atalho que podem ser usadas quando o Visual Studio estiver configurado para o padrão “Visual C# 2005”:

Descrição	Tecla de Atalho
Janela de propriedades do objeto	F4
Compilar	F6
Executar o programa	F5
Ir para o código fonte do formulário	F7
Retornar ao formulário	Shift + F7
Adicionar namespace	CTRL + .
Criar propriedade	CTRL + R + E
Depurar sem entrar nos métodos	F10
Depurar entrando nos métodos	F11
Adicionar Break Point	F9
Pesquisar pelo nome de um arquivo no projeto	CTRL + ,
Pesquisar no conteúdo de um arquivo	CTRL + F
Pesquisar no conteúdo de todos os arquivos do projeto	CTRL + SHIFT + F

Para alterar as configurações de teclado (Tools->Options):



Instalação do SQL SERVER – Configurando modo de autenticação Misto

Ao instalar o SQL Server, fique atento a este passo onde deve-ser marcar a opção Modo Misto caso queira se conectar utilizando um usuário do SQL Server, como por exemplo o “sa”.

The screenshot shows the 'Configuração do Mecanismo de Banco de Dados' (Configure the Database Engine) step in the SQL Server 2014 installation wizard. The window title is 'Instalação do SQL Server 2014'. The left sidebar lists various installation steps, with 'Configuração do Mecanismo d...' (Configure the Database Engine) currently selected. The main area has four tabs: 'Configuração do Servidor', 'Diretórios de Dados', 'Instâncias de Usuário', and 'FILESTREAM'. The 'Configuração do Servidor' tab is active, displaying options for authentication mode. The 'Modo de Autenticação' (Authentication Mode) section shows two radio buttons: 'Modo de Autenticação do Windows' (unselected) and 'Modo Misto (autenticação do SQL Server e do Windows)' (selected). Below this, there is a prompt to specify the password for the 'sa' (system administrator) account. Two password fields are shown: 'Digitar Senha:' and 'Confirmar senha:', both containing six dots. Below the password fields is a list box for 'Especificar administradores do SQL Server' (Specify SQL Server administrators), which currently contains 'PC_EDU\Eduardo (Eduardo)'. To the right of the list box is a note: 'Os administradores do SQL Server têm acesso irrestrito ao Mecanismo de Banco de Dados.' (SQL Server administrators have unrestricted access to the Database Engine). At the bottom of the list box are three buttons: 'Adicionar Usuário Atual', 'Adicionar', and 'Remover'. At the very bottom of the wizard window are four buttons: '< Voltar', 'Avançar >', 'Cancelar', and 'Ajuda'.

Instalação do SQL Server 2014

Configuração do Mecanismo de Banco de Dados

Especifique o modo de segurança da autenticação, os administradores e os diretórios de dados do Mecanismo de Banco de Dados.

Regras Globais
Microsoft Update
Atualizações de Produto
Instalar Arquivos de Instalação
Instalar Regras
Termos de Licença
Seleção de Recursos
Regras de Recurso
Configuração da Instância
Configuração do Servidor
Configuração do Mecanismo d...
Regras de Configuração de Rec...
Andamento da Instalação
Concluída

Configuração do Servidor | Diretórios de Dados | Instâncias de Usuário | FILESTREAM

Especifique o modo de autenticação e os administradores para o Mecanismo de Banco de Dados.

Modo de Autenticação

☐ Modo de Autenticação do Windows

☒ **Modo Misto** (autenticação do SQL Server e do Windows)

Especifique a senha da conta do sa (administrador do sistema) do SQL Server.

Digitar Senha: [password field]

Confirmar senha: [password field]

Especificar administradores do SQL Server

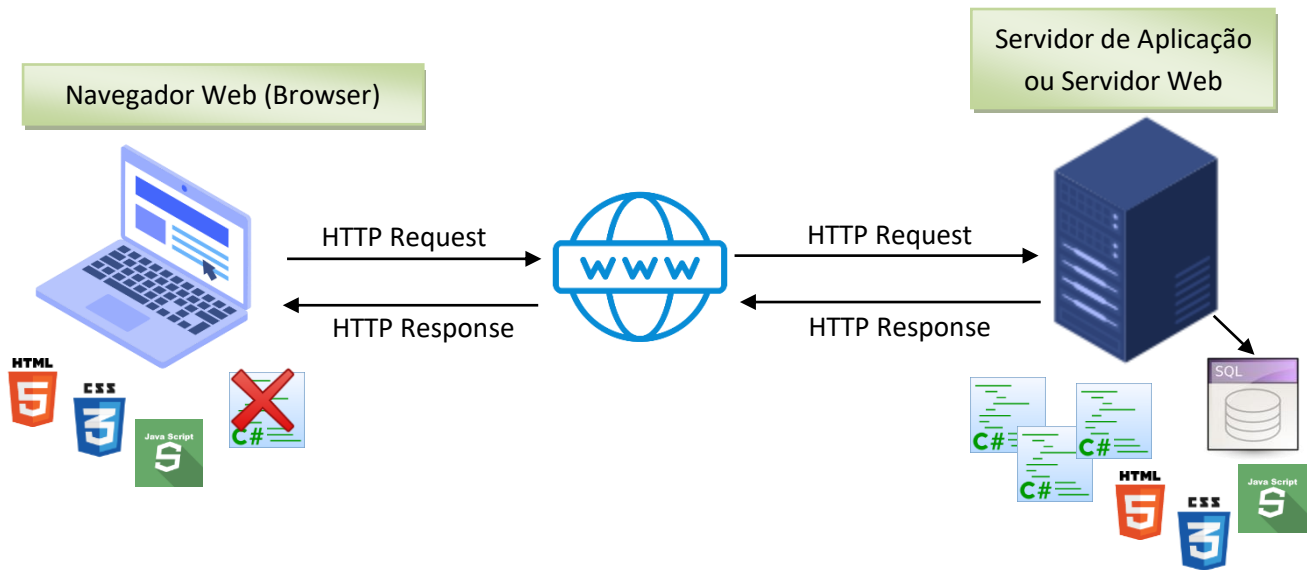
PC_EDU\Eduardo (Eduardo)

Os administradores do SQL Server têm acesso irrestrito ao Mecanismo de Banco de Dados.

Adicionar Usuário Atual | Adicionar | Remover

< Voltar | Avançar > | Cancelar | Ajuda

1. Arquitetura de uma aplicação WEB



“Servidor web é um software responsável por aceitar pedidos em HTTP de clientes, geralmente os navegadores, e servi-los com respostas em HTTP, incluindo opcionalmente dados, que geralmente são páginas web, tais como documentos em HTML com objetos embutidos (imagens, etc) ou um computador que executa um programa que provê a funcionalidade descrita anteriormente.[1] O mais popular, e mais utilizado no mundo, é o servidor Apache (software livre). A Microsoft possui a sua própria solução denominada IIS (Internet Information Services).”

Fonte: https://pt.wikipedia.org/wiki/Servidor_web

2. HTML

Vide Apostila Caelum

3. CSS

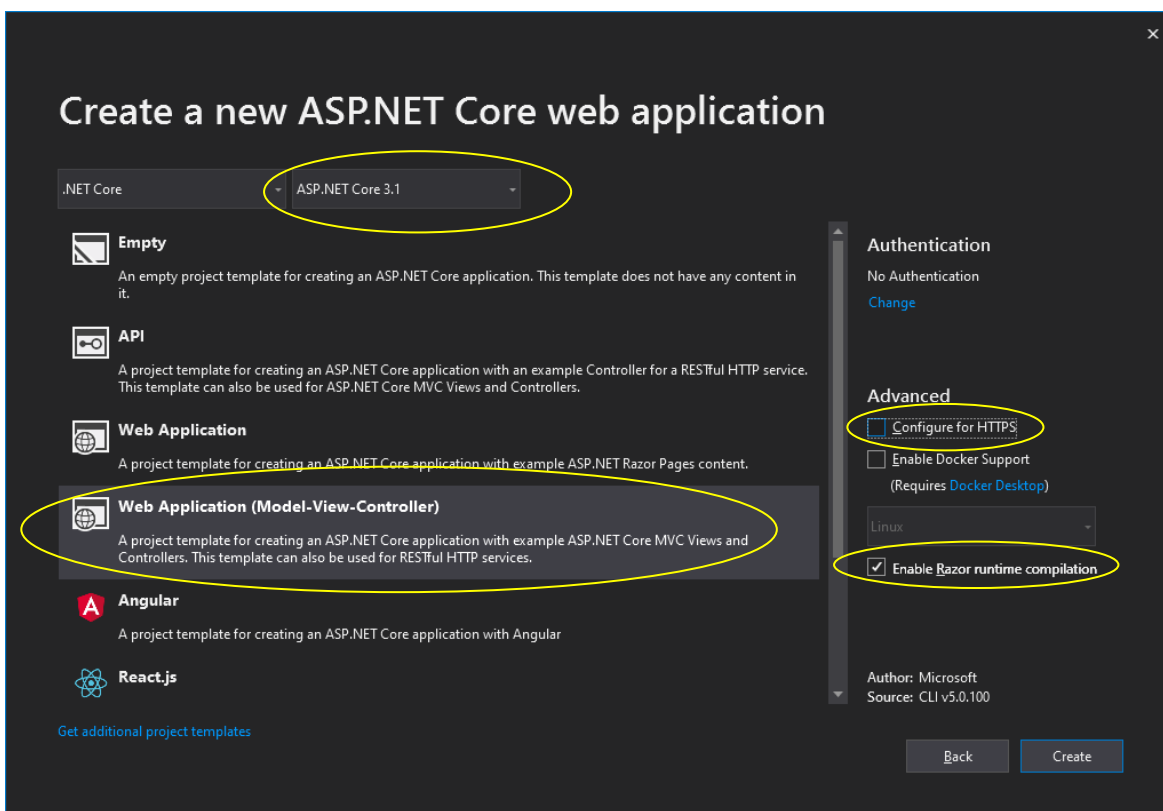
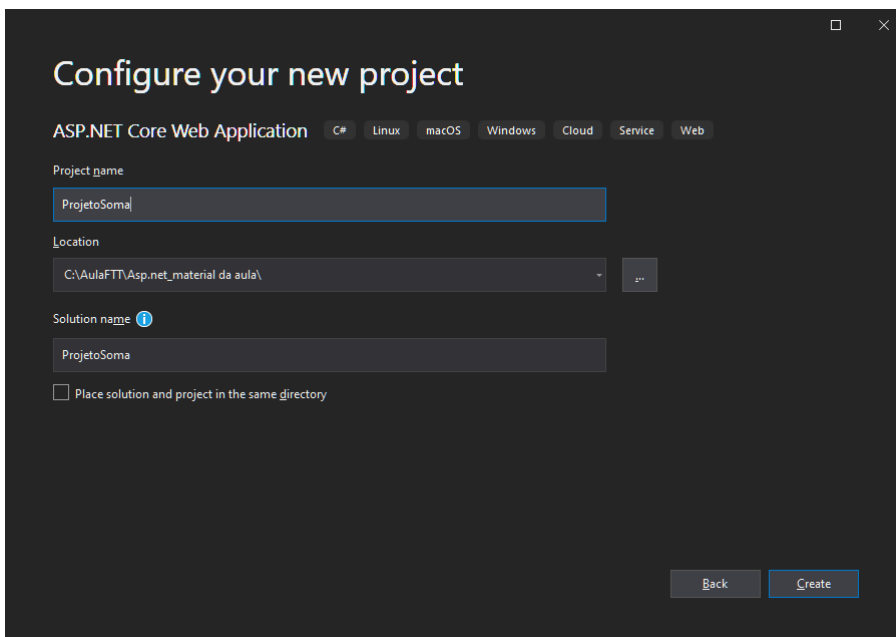
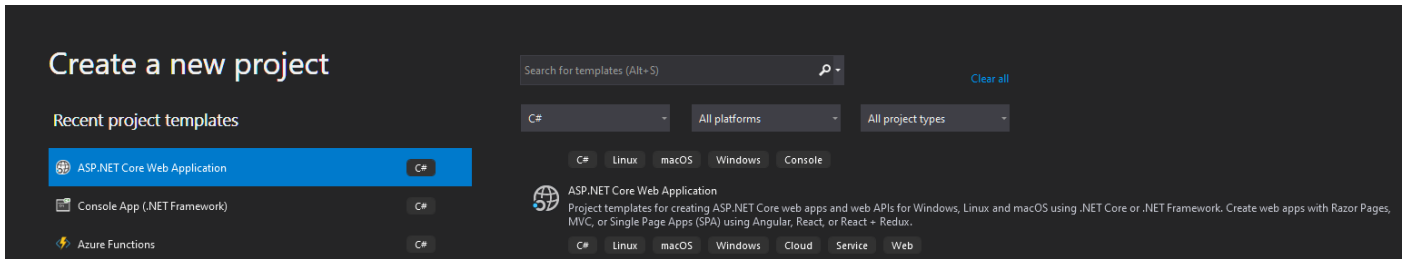
Vide Apostila Caelum

4. Javascript

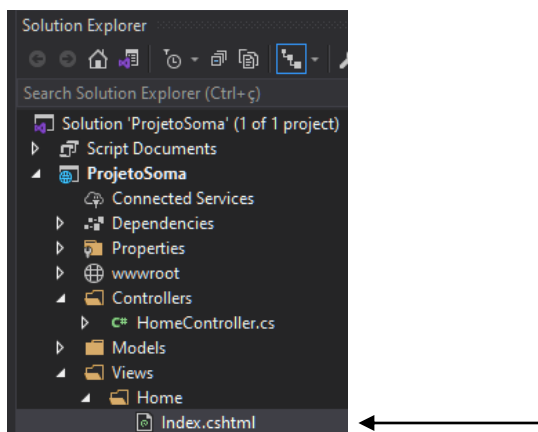
Vide Apostila Caelum

5. Criando o primeiro projeto Asp.Net Core – Projeto soma

Vá em File-> New Project



Editar o arquivo Index.cshtml



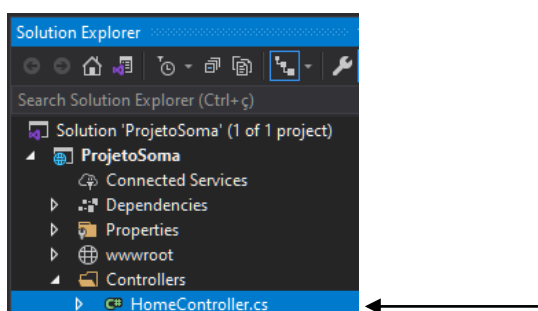
Apague o conteúdo que está no arquivo, e deixe-o como abaixo:

```
<form asp-action="EfetuaSoma">

    Valor 1:
    <input type="number" name="valor1" />
    <br />
    Valor 2:
    <input type="number" name="valor2" />
    <br>
    <input type="submit" value="Somar" />
    <br />
    <br />
    <br />
    Resultado:
    <input type="number" value="@Model" disabled />

</form>
```

Agora, edite o arquivo HomeController.cs, inserindo o método abaixo:



```
public IActionResult EfetuaSoma(int valor1, int valor2)
{
    int resultado = valor1 + valor2;
    return View("Index", resultado);
}
```

Execute o projeto.

6. Criando um projeto para o cálculo de tabuada

Vamos agora criar um novo projeto para adicionar o cálculo de tabuada ao projeto existente. Crie o mesmo projeto como explicado no capítulo anterior.

Editar o arquivo `Index.cshtml` apagando o conteúdo do arquivo e insira o seguinte conteúdo:

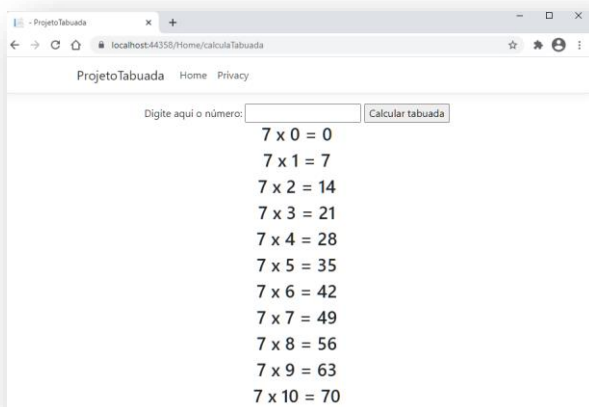
```
<div class="text-center">
  <form asp-action="calculaTabuada">
    Digite aqui o número:
    <input type="number" name="numero" />
    <input type="submit" value="Calcular tabuada" />
  </form>

  @if (Model > 0)
  {
    @for (int i = 0; i <= 10; i++)
    {
      <h3>@Model x @i = @Model * i</h3> <BR>
    }
  }
</div>
```

Agora, edite o arquivo `HomeController.cs`, inserindo o método abaixo:

```
public IActionResult CalculaTabuada(int numero)
{
    return View("Index", numero);
}
```

Execute o projeto:

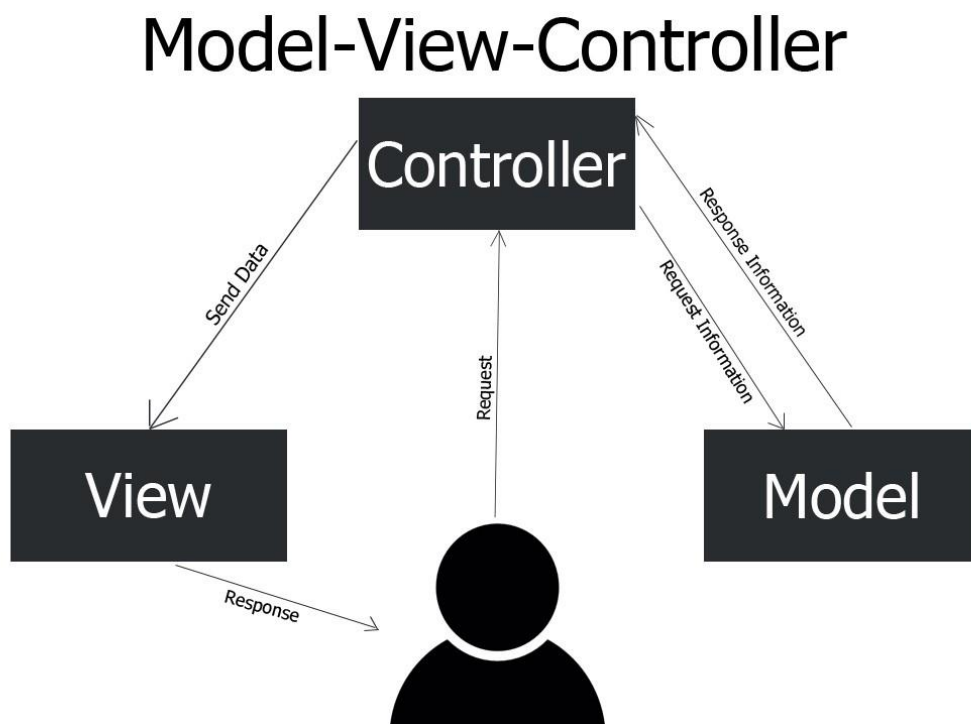


Exercício

- 6.1 Crie um projeto para calcular a potência. O usuário deverá informar a base e o expoente e o sistema deverá retornar o resultado na página HTML.
- 6.2 Crie um novo projeto para um conversor de medidas. Disponibilize uma caixa de texto para o usuário digitar o valor (sempre em metros) e em uma caixa combo ele deverá informar uma unidade de medida para a qual deseja converter o valor em metros. As unidades disponíveis serão: centímetro, milímetro, quilômetro, pé e milha. Efetue a conversão e exiba o resultado. Dica: Para caixa combo, pesquise pela tag `<select>` :)

7. Arquitetura Model View Controller

Detalhes: <https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>



[1]

Model: The model represents a data structure, a representation of the domain that we are trying to model.

View: The view's responsibility is to present a model to a user, in our case, as a web user interface, so mainly HTML, CSS, and JavaScript.

Controller: The controller is the key component of MVC. It plays the coordinator role between a request from a user to its response. The code of a controller should remain minimal and should not include complex logic or manipulation. The controller's primary responsibility is to handle a request and dispatch a response. The controller is an HTTP bridge.

8. Asp.net MVC Core

Introdução ao ASP.NET Core

<https://docs.microsoft.com/pt-br/aspnet/core/?view=aspnetcore-2.2>

Por Daniel Roth, Rick Anderson e Shaun Luttin

O ASP.NET Core é uma estrutura de software livre, de multiplataforma e alto desempenho para a criação de aplicativos modernos conectados à Internet e baseados em nuvem. Com o ASP.NET Core, você pode:

- Compilar aplicativos e serviços Web, aplicativos IoT e back-ends móveis.
- Usar suas ferramentas de desenvolvimento favoritas no Windows, macOS e Linux.

- Implantar na nuvem ou local.
- Executar no .NET Core ou no .NET Framework.

Por que escolher o ASP.NET Core?

Milhões de desenvolvedores usaram (e continuam usando) o ASP.NET 4.x para criar aplicativos Web. O ASP.NET Core é uma reformulação do ASP.NET 4.x, com alterações de arquitetura que resultam em uma estrutura mais enxuta e modular.

O ASP.NET Core oferece os seguintes benefícios:

- Uma história unificada para a criação da interface do usuário da Web e das APIs Web.
- Projetado para capacidade de teste.
- O Razor Pages torna a codificação de cenários focados em página mais fácil e produtiva.
- **O Blazor permite que você use C# no navegador junto com o JavaScript. Compartilhe a lógica de aplicativo do lado do cliente e do servidor toda escrita com o .NET.**
- Capacidade de desenvolver e executar no Windows, macOS e Linux.
- De software livre e voltado para a comunidade.
- Integração de estruturas modernas do lado do cliente e fluxos de trabalho de desenvolvimento.
- Um sistema de configuração pronto para a nuvem, baseado no ambiente.
- Injeção de dependência interna.
- Um pipeline de solicitação HTTP leve, modular e de alto desempenho.
- Capacidade de hospedar o seguinte:
 - Kestrel
 - IIS
 - HTTP.sys
 - Nginx
 - Apache
 - Docker
- Controle de versão lado a lado.
- Ferramentas que simplificam o moderno desenvolvimento para a Web.

9. Acesso ao banco de dados com C#

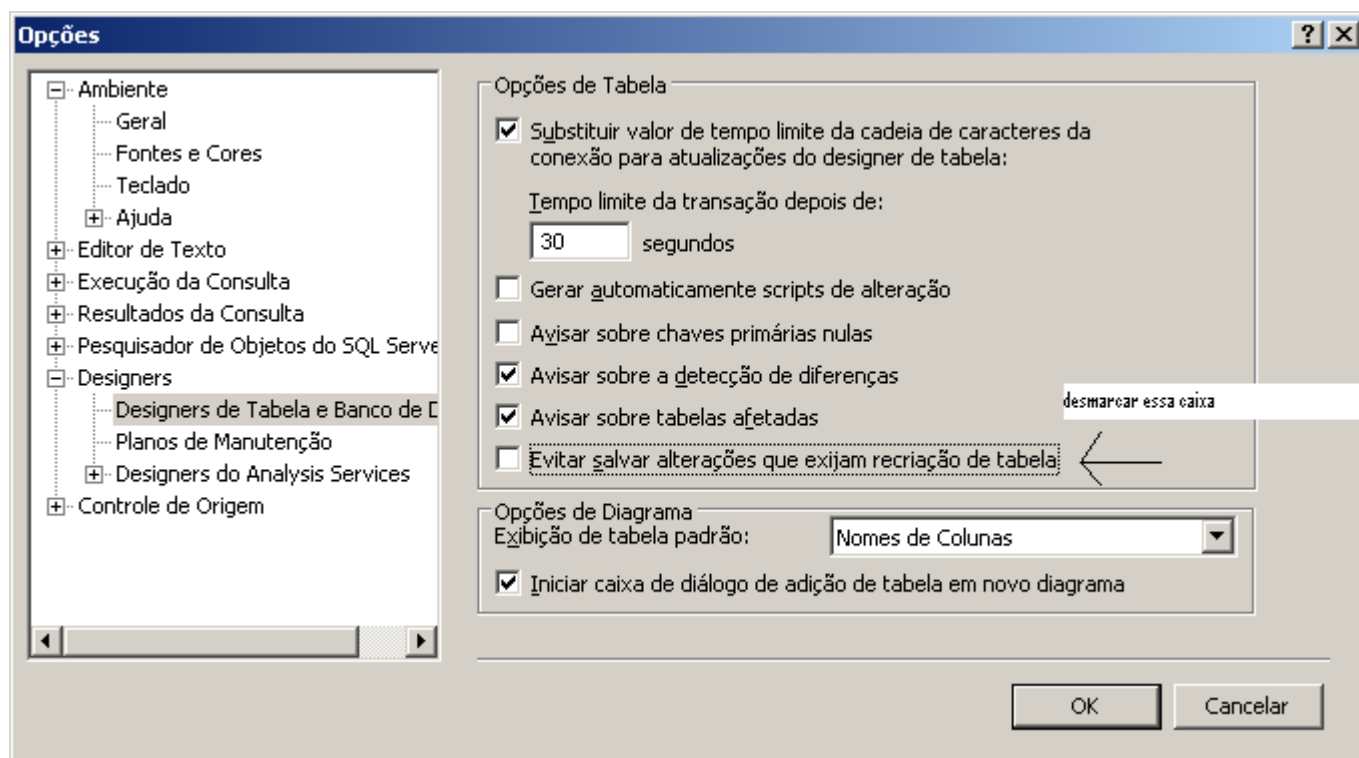
No SQL Server, crie um novo banco de dados chamado AulaDB.



Rode a instrução SQL para criar a tabela aluno no banco de dados criado acima.

```
CREATE TABLE Alunos (
    Id int NOT NULL PRIMARY KEY,
    nome varchar(50) NULL,
    mensalidade decimal (18, 2) NULL,
    cidadeId int NULL,
    DataNascimento datetime NULL
)
```

Caso tente alterar a estrutura de uma tabela no modo visual, o **SQL Server** pode dar um erro dizendo que não é possível alterar a estrutura da tabela. Para permitir essa alteração, entre nas opções do SQL Server e desmarque a seguinte opção:



Projeto Windows Forms

A fim de focarmos apenas na parte de acesso ao banco de dados, iremos atuar inicialmente em um projeto Windows Forms. Na sequência iremos aplicar estes mesmos conceitos de acesso ao banco em um projeto web usando para isso o ASP.NET Core.

Sendo assim, crie um projeto Windows forms.

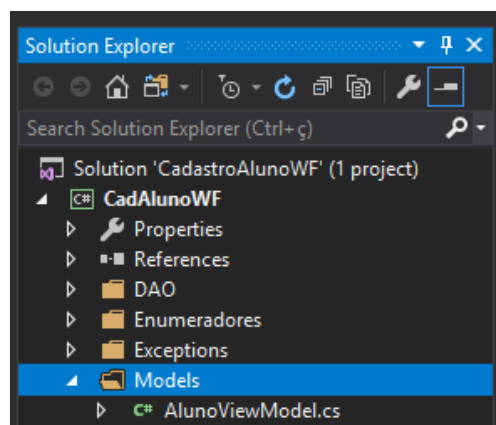
Classes para transferência de dados entre as camadas da aplicação: ViewModel ou VO (Value Object) ou DTO (Data Transfer Object)

Referências:

<https://www.oracle.com/java/technologies/transfer-object.html>

Esta classe facilita o transporte das informações entre as camadas da aplicação. Trata-se da classe que irá conter todos os atributos da tabela Aluno. Com o objetivo de encapsular os atributos, podemos utilizar métodos de acesso públicos (Getters e Setters) para cada atributo.

Crie uma pasta chamada Models no projeto e dentro desta pasta crie a classe AlunoViewModel:

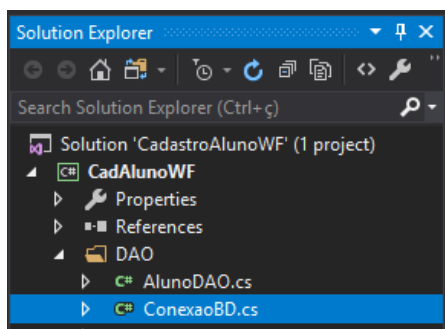


```
using System;
namespace CadAlunoWF.Models
{
    public class AlunoViewModel
    {
        public int Id { get; set; }
        public string Nome { get; set; }
        public double Mensalidade { get; set; }
        public int CidadeId { get; set; }
        public DateTime DataNascimento { get; set; }
    }
}
```

Classe de Conexão com o Banco de dados

A aplicação deve ter apenas uma classe de conexão com banco de dados. Sempre que for necessário utilizar uma conexão, devemos acessar essa classe e executar o método que devolve a conexão aberta. Para se conectar ao banco de dados é necessária uma string de conexão. Esta classe implementa o design pattern Factory que prega o encapsulamento da construção (fabricação) de objetos complicados.

Crie uma pasta chamada "DAO" e crie uma classe chamada "ConexaoBD"



```
using System.Data.SqlClient;

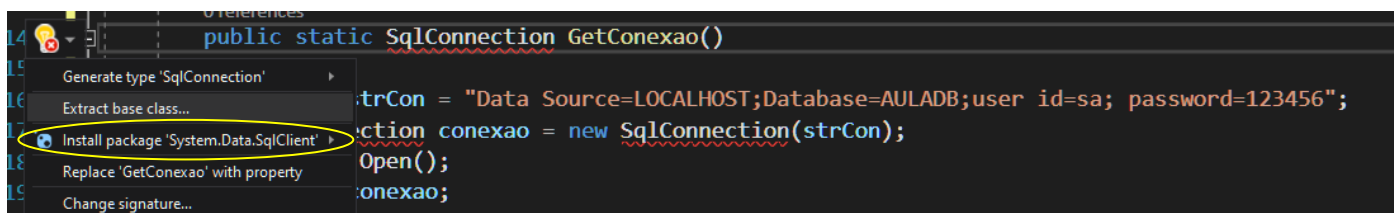
namespace DAO
{
    public static class ConexaoBD
    {
        /// <summary>
        /// Método Estático que retorna um conexão aberta com o BD
        /// </summary>
        /// <returns>Conexão aberta</returns>
        public static SqlConnection GetConexao()
        {
            string strCon = "Data Source=LOCALHOST; Database=AULADB; user id=sa; password=123456";
            SqlConnection conexao = new SqlConnection(strCon);
            conexao.Open();
            return conexao;
        }
    }
}
```

Caso não saiba a senha, tente se conectar por autenticação Windows. A string neste caso fica assim:

```
"Data Source=LOCALHOST; Database=AULADB; integrated security=true";
```

Obs: Quando estiver utilizando o **.NET Core 3.1**, será necessário adicionar a biblioteca **SqlClient**. Para tanto proceda da seguinte forma:

1 – Clique sobre a classe `SqlConnection` e pressione control + ponto (.). Selecione a opção indicada:



Classe DAO para realização das operações de acesso ao banco de dados

Referências:

- [1] <https://www.oracle.com/java/technologies/dataaccessobject.html>
- [2] http://www.macoratti.net/11/10/pp_dao1.htm

DAO (acrônimo de Data Access Object), é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados. O padrão DAO consiste em abstrair o mecanismo de persistência utilizado na aplicação. A camada de negócios acessa os dados persistidos sem ter conhecimento se os dados estão em um banco de dados relacional ou um arquivo XML. O padrão DAO esconde os detalhes da execução da origem dos dados.

Este padrão permite criar as classes de dados independentemente da fonte de dados ser um BD relacional, um arquivo texto, um arquivo XML, etc. Para isso, ele encapsula os mecanismos de acesso a dados e cria uma interface de cliente genérica para fazer o acesso aos dados permitindo que os mecanismos de acesso a dados sejam alterados independentemente do código que utiliza os dados. [2]

Existem diversas implementações do padrão DAO mas em geral podemos relacionar algumas características desejáveis em uma implementação do padrão DAO:

- Todo o acesso aos dados deve ser feita através das classes DAO de forma a se ter o encapsulamento;
- Cada instância da DAO é responsável por um objeto de domínio;
- O DAO deve ser responsável pelas operações CRUD no domínio;
- O DAO não deve ser responsável por transações, sessões ou conexões que devem ser tratados fora do DAO;

Por enquanto iremos incluir apenas o método para incluir alunos:

Na pasta DAO, crie a classe AlunoDAO

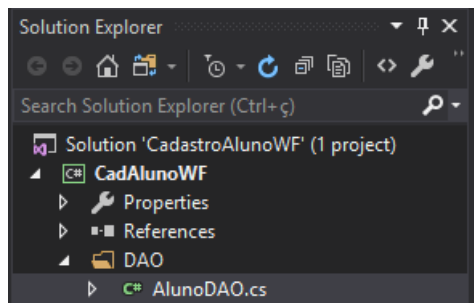
```
using CadAlunoWF.Models;
using System;
using System.Data.SqlClient;

namespace DAO
{
    public class AlunoDAO
    {
        /// <summary>
        /// Método para inserir um aluno no BD
        /// </summary>
        /// <param name="aluno">objeto aluno com todas os atributos preenchidos</param>
        public void Inserir(AlunoViewModel aluno)
        {
            SqlConnection conexao = ConexaoBD.GetConexao();
            try
            {
                //devemos substituir a ',' por '.'
                string mensalidade = aluno.Mensalidade.ToString().Replace(',', '.');
                // set dateformat dmy; este comando serve para alterar a
                //forma como o SQL Server entende o formato de data
                string sql = String.Format("set dateformat dmy; " +
                    "insert into alunos(id, nome, mensalidade, cidadeId, dataNascimento)" +
                    "values ( {0}, '{1}', {2}, {3}, '{4}')" , aluno.Id,
                    aluno.Nome, mensalidade, aluno.CidadeId, aluno.DataNascimento);
                SqlCommand comando = new SqlCommand(sql, conexao);
                comando.ExecuteNonQuery();
            }
        }
    }
}
```

```

        finally
        {
            conexao.Close();
        }
    }
}

```



Formulário de Cadastro

O formulário para incluir terá esta aparência:

Método de validação e código do botão inserir:

```

public void ValidaAluno(AlunoViewModel a)
{
    if (a.Id < 0)
        throw new Exception("Id não pode ser negativo!");

    if (string.IsNullOrEmpty(a.Nome))
        throw new Exception("Informe o nome!");

    if (a.Mensalidade < 0)
        throw new Exception("Mensalidade não pode ser negativa!");

    if (a.CidadeId <= 0)
        throw new Exception("Código da cidade não ser negativo!");

    if (a.DataNascimento > DateTime.Now)
        throw new Exception("Data de nascimento inválida!");
}

```



```

private void btnInserir_Click(object sender, EventArgs e)
{
    try
    {
        AlunoViewModel aluno = PreencheDadosVO();
        ValidaAluno(aluno);
        AlunoDAO dao = new AlunoDAO();
        dao.Inserir(aluno);
    }
    catch (Exception erro)
    {
        MessageBox.Show(erro.Message);
    }
}

/// <summary>
/// Preenche os dados com base nos campos da tela
/// </summary>
/// <returns>objeto AlunoVO</returns>
private AlunoViewModel PreencheDadosVO()
{
    AlunoViewModel aluno = new AlunoViewModel();
    aluno.Id = Convert.ToInt32(txtId.Text);
    aluno.CidadeId = Convert.ToInt32(txtCidadeId.Text);
    aluno.Mensalidade = Convert.ToDouble(txtMensalidade.Text);
    aluno.Nome = txtNome.Text;
    aluno.DataNascimento = Convert.ToDateTime(txtData.Text);
    return aluno;
}

```

Exercícios:

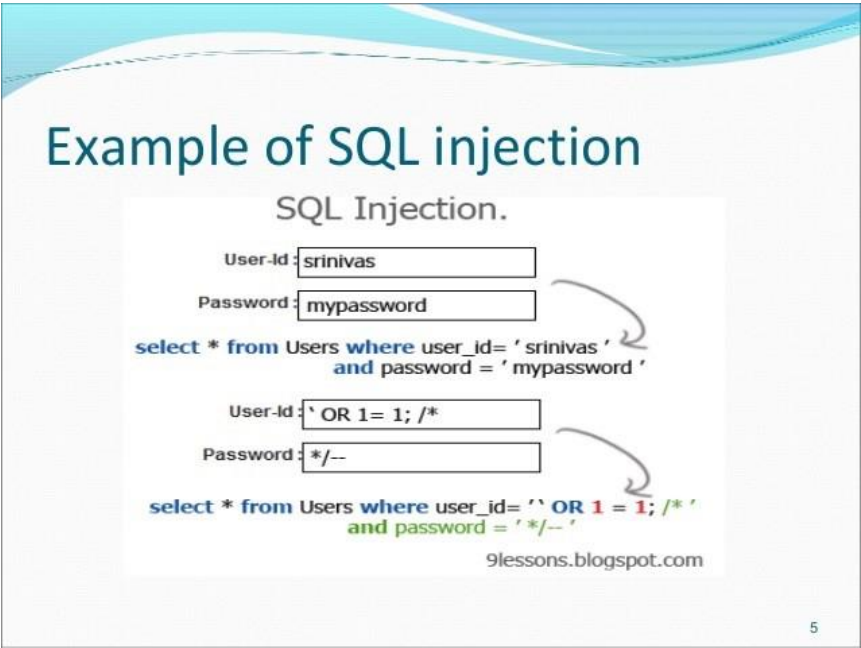
- 9.1 Adicione na classe DAO os métodos para alterar e excluir um aluno. Adicione os botões na tela para que seja possível efetuar estes novos dois métodos.

10. SQL Injection

A Injeção de SQL, mais conhecida através do termo americano SQL Injection, é um tipo de ameaça de segurança que se aproveita de falhas em sistemas que interagem com bases de dados via SQL. A injeção de SQL ocorre quando o atacante consegue inserir uma série de instruções SQL dentro de uma consulta (query) através da manipulação das entradas de dados de uma aplicação.

Fontes:

https://pt.wikipedia.org/wiki/Inje%C3%A7%C3%A3o_de_SQL
<https://www.tecmundo.com.br/tecmundo-explica/113195-sql-injection-saiba-tudo-ataque-simples-devastador.htm>



Prevenindo o SQL Injection no C#:

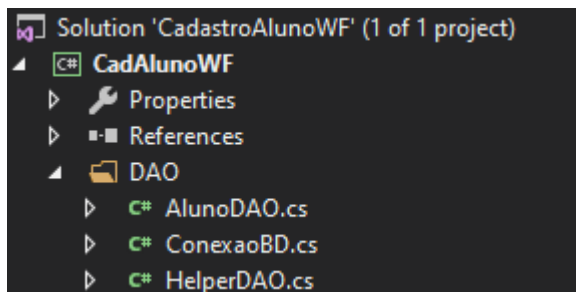
Fontes:

<https://docs.microsoft.com/en-us/archive/msdn-magazine/2004/september/data-security-stop-sql-injection-attacks-before-they-stop-you>
<https://www.devmedia.com.br/prevenindo-sql-injection-asp-net/17545>
<https://pt.stackoverflow.com/questions/100729/como-acontece-um-sql-injection>

Principle	Implementation	
Never trust user input	Validate all textbox entries using validation controls, regular expressions, code, and so on	<p>All Input is Evil</p> <p>The first principle listed in Figure 4 is extremely important: assume that all user input is evil! You should never use unvalidated user input in a database query. The ASP.NET validation controls—especially the <code>RegularExpressionValidator</code> control—are a good tool for validating user input.</p>
Never use dynamic SQL	Use parameterized SQL or stored procedures	
Never connect to a database using an admin-level account	Use a limited access account to connect to the database	
Don't store secrets in plain text	Encrypt or hash passwords and other sensitive data; you should also encrypt connection strings	
Exceptions should divulge minimal information	Don't reveal too much information in error messages; use <code>customErrors</code> to display minimal information in the event of unhandled error; set <code>debug</code> to false	

11. Evitando SQL Injection - Ajustes no cadastro de alunos

Primeiro, vamos criar uma classe estática chamada **HelperDAO** para adicionar alguns métodos que utilizaremos em várias classes DAO. O bloco “using” faz o “dispose” dos objetos, garantindo assim que o mesmo seja destruído, fazendo às vezes do bloco try-finally. Adicione esta classe na pasta **DAO**:



```
using System.Data;
using System.Data.SqlClient;
using System.Linq;
```

```
namespace CadAlunoWF.DAO
{
    public static class HelperDAO
    {
        public static void ExecutaSQL(string sql, SqlParameter[] parametros)
        {
            using (SqlConnection conexao = ConexaoBD.GetConexao())
            {
                using (SqlCommand comando = new SqlCommand(sql, conexao))
                {
                    if (parametros != null)
                        comando.Parameters.AddRange(parametros);
                    comando.ExecuteNonQuery();
                }
                conexao.Close();
            }
        }
    }
}
```

Classe DAO com parâmetros. Métodos excluir e alterar registros

Nova Classe DAO, desta vez utilizando parâmetros para enviar os dados para o banco, evitando assim o SQL Injection. Segue abaixo a classe completa, com os métodos de inclusão, alteração e exclusão.

```
using System.Data.SqlClient;
using CadAlunoWF.Models;

namespace CadAlunoWF.DAO
{
    public class AlunoDAO
    {
        private SqlParameter[] CriaParametros(AlunoViewModel aluno)
        {
            SqlParameter[] parametros = new SqlParameter[5];
            parametros[0] = new SqlParameter("id", aluno.Id);
            parametros[1] = new SqlParameter("nome", aluno.Nome);
            parametros[2] = new SqlParameter("mensalidade", aluno.Mensalidade);
            parametros[3] = new SqlParameter("cidadeId", aluno.CidadeId);
            parametros[4] = new SqlParameter("dataNascimento", aluno.DataNascimento);
            return parametros;
        }

        /// <summary>
        /// Método para inserir um aluno no BD
        /// </summary>
        /// <param name="aluno">objeto aluno com todas os atributos preenchidos</param>
        public void Inserir(AlunoViewModel aluno)
        {
            string sql =
                "insert into alunos(id, nome, mensalidade, cidadeId, dataNascimento)" +
                "values ( @id, @nome, @mensalidade, @cidadeId, @dataNascimento)";
            HelperDAO.ExecutaSQL(sql, CriaParametros(aluno));
        }

        /// <summary>
        /// Altera um aluno no banco de dados
        /// </summary>
        /// <param name="aluno">objeto aluno com todas os atributos preenchidos</param>
        public void Alterar(AlunoViewModel aluno)
        {
            string sql =
                "update alunos set nome=@nome, mensalidade=@mensalidade, " +
                "cidadeId=@cidadeId, dataNascimento=@dataNascimento where id = @id";
            HelperDAO.ExecutaSQL(sql, CriaParametros(aluno));
        }

        /// <summary>
        /// Exclui um aluno no banco de dados.
        /// </summary>
        /// <param name="id">id do aluno</param>
        public void Excluir(int id)
        {
            string sql = "delete alunos where id =" + id;
            HelperDAO.ExecutaSQL(sql, null);
        }
    }
}
```

Alterações no formulário para excluir e alterar

Para testar estes novos dois métodos, vamos alterar o formulário principal, incluindo 2 novos botões:

Código dos botões alterar e excluir:

```
private void btnAlterar_Click(object sender, EventArgs e)
{
    try
    {
        AlunoViewModel aluno = PreencheDadosVO();
        ValidaAluno(aluno);
        AlunoDAO dao = new AlunoDAO();
        dao.Alterar(aluno);
    }
    catch (Exception erro)
    {
        MessageBox.Show(erro.Message);
    }
}

private void btnExcluir_Click(object sender, EventArgs e)
{
    try
    {
        AlunoDAO dao = new AlunoDAO();
        dao.Excluir(Convert.ToInt32(txtId.Text));
    }
    catch (Exception erro)
    {
        MessageBox.Show(erro.Message);
    }
}
```

Exercícios:

11.1 - Em uma nova solution, crie um cadastro de jogos (inclusão, alteração e exclusão), utilizando os novos conceitos aprendidos.

Tabela:

```
CREATE TABLE jogos(
    [id] [int] NOT NULL primary key,
    [descricao] [varchar](50) NULL,
    [valor_locacao] [decimal](18, 2) NULL,
    [data_aquisicao] [datetime] NULL,
    [categoriaID] [int] NULL)
```

12. Consultas

Para armazenarmos o conteúdo consultado de uma tabela, iremos utilizar um objeto da classe DataTable. Esta classe permite criar uma tabela “virtual” (apenas em memória), onde são armazenados os registros de uma tabela consultada.

Adicione na classe DAO o seguintes métodos:

```

/// <summary>
/// Consulta um aluno com base eu seu id
/// </summary>
/// <param name="id">id do aluno</param>
/// <returns></returns>
public AlunoViewModel Consulta(int id)
{
    using (SqlConnection cx = ConexaoBD.GetConexao())
    {
        string sql = "select * from alunos where id = " + id;
        using (SqlDataAdapter adapter = new SqlDataAdapter(sql, cx))
        {
            DataTable tabela = new DataTable();
            adapter.Fill(tabela);
            cx.Close();
            if (tabela.Rows.Count == 0)
                return null;
            else
            {
                DataRow registro = tabela.Rows[0];
                return MontaModel(registro);
            }
        }
    }
}

/// <summary>
/// Recebe uma registro e e preenche um objeto AlunoVO
/// </summary>
/// <param name="registro">1 registro (linha) do DataTable</param>
/// <returns>Objeto com os atributos preenchidos</returns>
public static AlunoViewModel MontaModel(DataRow registro)
{
    AlunoViewModel aluno = new AlunoViewModel();

    aluno.Id = Convert.ToInt32(registro["id"]);
    aluno.Nome = registro["nome"].ToString();
    aluno.CidadeId = Convert.ToInt32(registro["cidadeId"]);
    aluno.Mensalidade = Convert.ToDouble(registro["mensalidade"]);
    aluno.DataNascimento = Convert.ToDateTime(registro["DataNascimento"]);

    return aluno;
}

```

No formulário, adicione um botão para efetuar a consulta e coloque o seguinte código:

```

private void PreencheTela(AlunoViewModel a)
{
    if (a != null)
    {
        txtId.Text = a.Id.ToString();
        txtNome.Text = a.Nome;
        txtCidadeId.Text = a.CidadeId.ToString();
        txtData.Text = a.DataNascimento.ToShortDateString();
        txtMensalidade.Text = a.Mensalidade.ToString();
    }
}

private void BtnConsulta_Click(object sender, EventArgs e)
{
    try
    {
        AlunoDAO dao = new AlunoDAO();
        AlunoViewModel a = dao.Consulta(Convert.ToInt32(txtId.Text));
        if (a != null)
            PreencheTela(a);
        else
            MessageBox.Show("Registro não encontrado!");
    }
    catch (Exception erro)
    {
        MessageBox.Show(erro.Message);
    }
}

```

Redução de código

No intuito de evitar a redundância de código, iremos criar o método para executar a instrução SQL da consulta na classe **HelperDAO**:

Obs: será necessário adicionar o namespace `using System.Data;`

```

/// <summary>
/// Executa uma instrução Select
/// </summary>
/// <param name="sql">instrução SQL</param>
/// <returns>DataTable com os dados da instrução SQL</returns>
public static DataTable ExecutaSelect(string sql, SqlParameter[] parametros)
{
    using (SqlConnection conexao = ConexaoBD.GetConexao())
    {
        using (SqlDataAdapter adapter = new SqlDataAdapter(sql, conexao))
        {
            if (parametros != null)
                adapter.SelectCommand.Parameters.AddRange(parametros);

            DataTable tabelaTemp = new DataTable();
            adapter.Fill(tabelaTemp);
            conexao.Close();
            return tabelaTemp;
        }
    }
}

```

Por conta deste método, faça a seguinte alteração no método consulta da classe DAO:

```

/// <summary>
/// Consulta um aluno com base eu seu id
/// </summary>
/// <param name="id">id do aluno</param>
/// <returns></returns>
public AlunoViewModel Consulta(int id)
{
    string sql = "select * from alunos where id = " + id;
    DataTable tabela = HelperDAO.ExecutaSelect(sql, null);
    if (tabela.Rows.Count == 0)
        return null;
    else
        return MontaModel(tabela.Rows[0]);
}

```

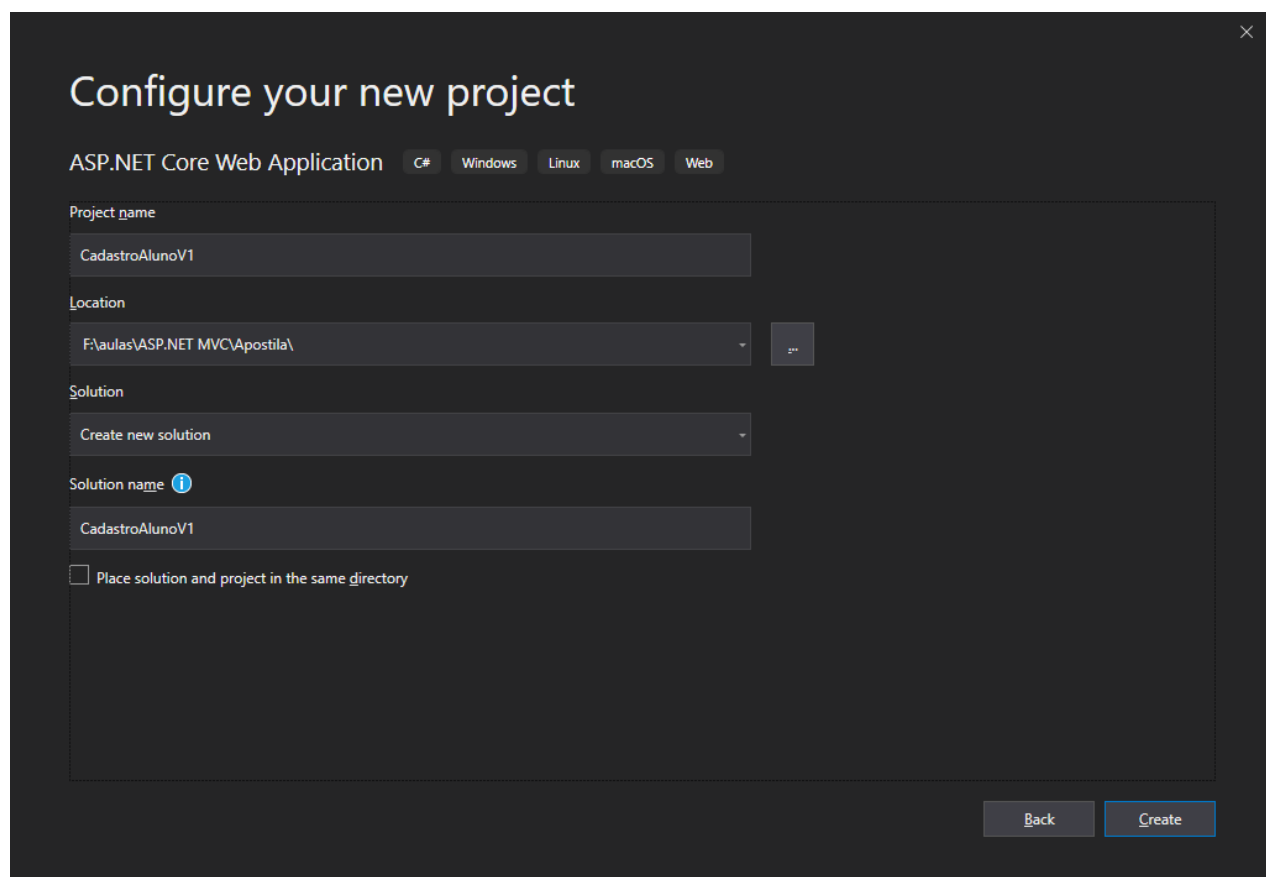
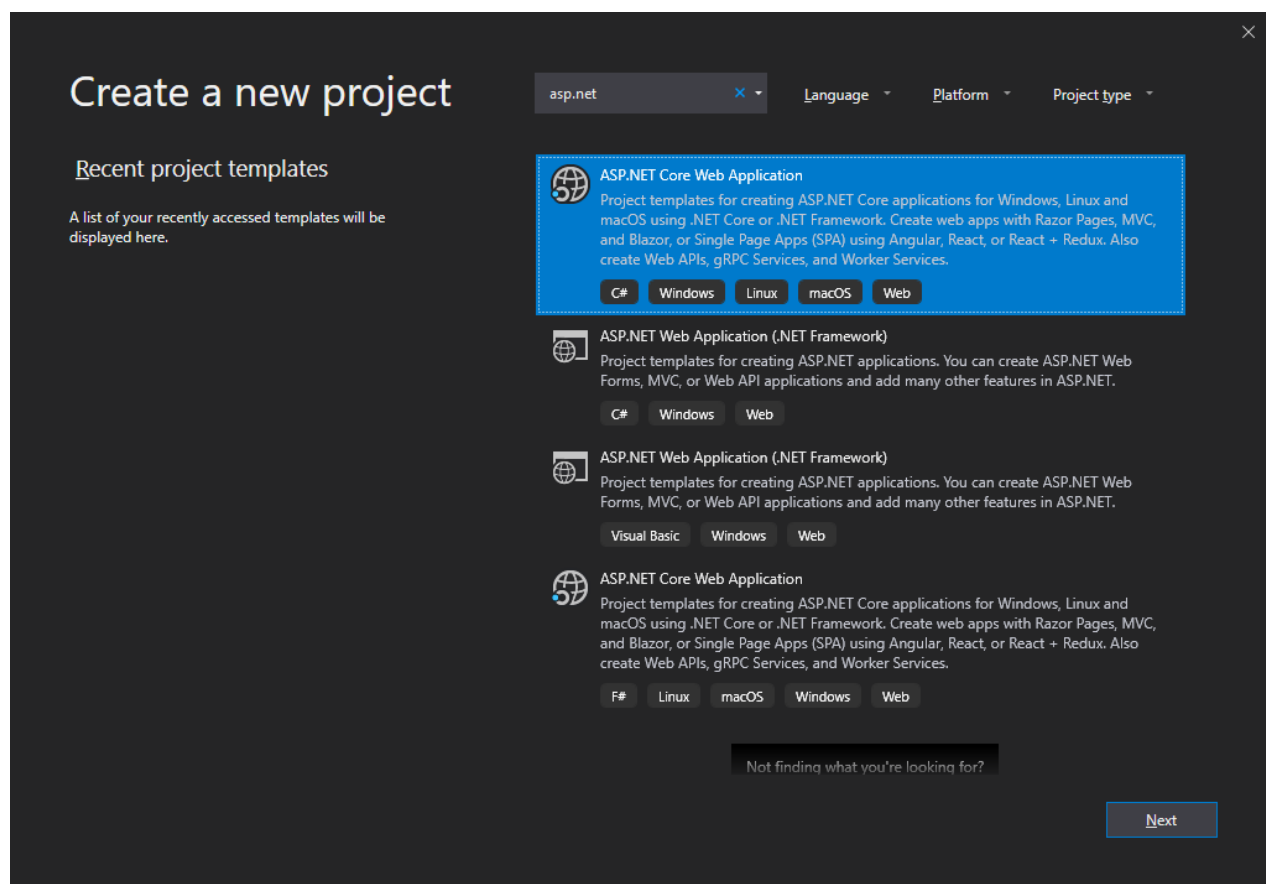
Exercícios:

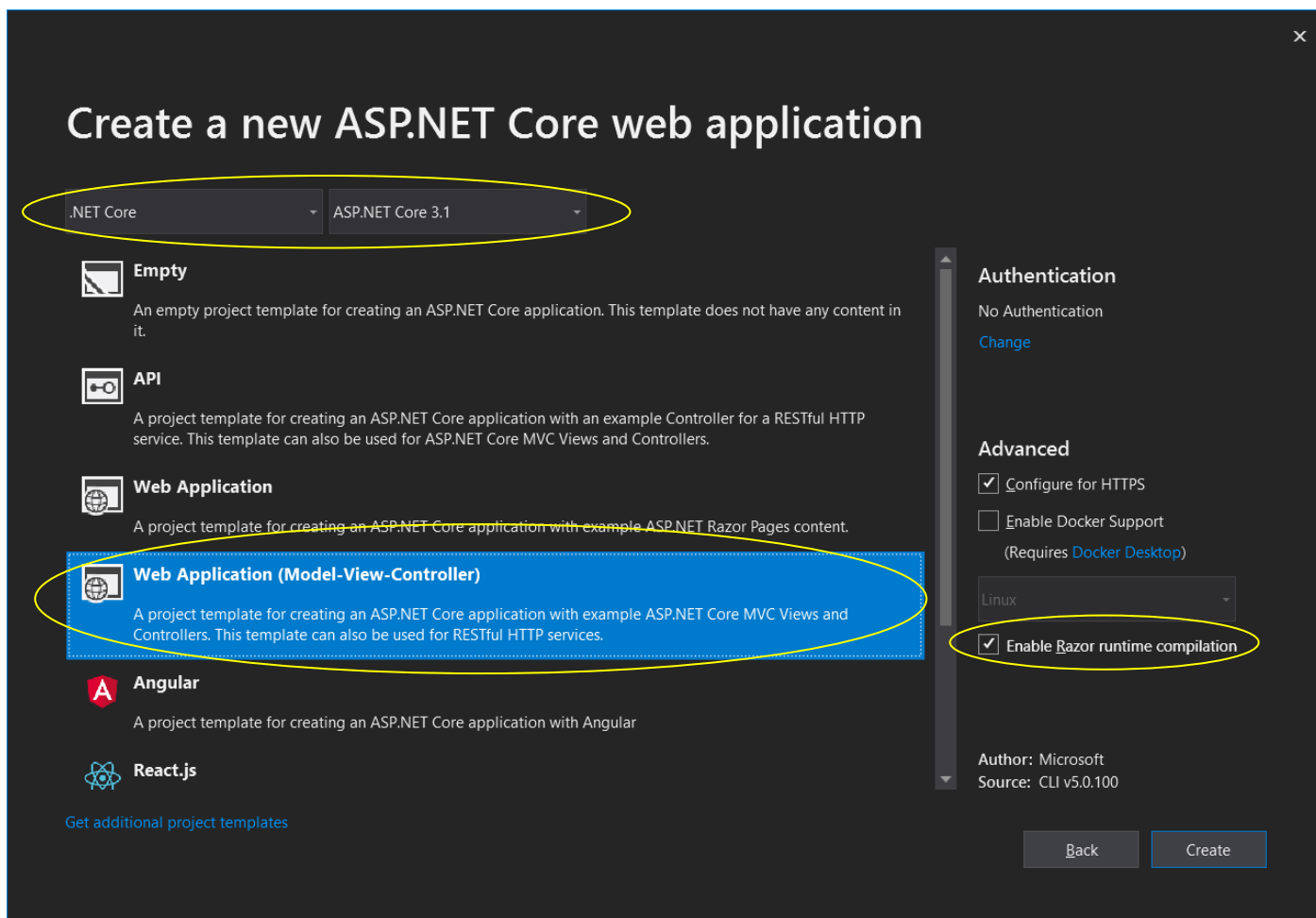
12.1) Faça a consulta no cadastro de jogos, aplicando também as reduções no código.

12.2) Faça no cadastro de jogos um método para retornar todos os registros de jogos (List<JogoViewModel>), exibindo-os em algum componente, como Listbox, Textbox ou mesmo um Gridview.

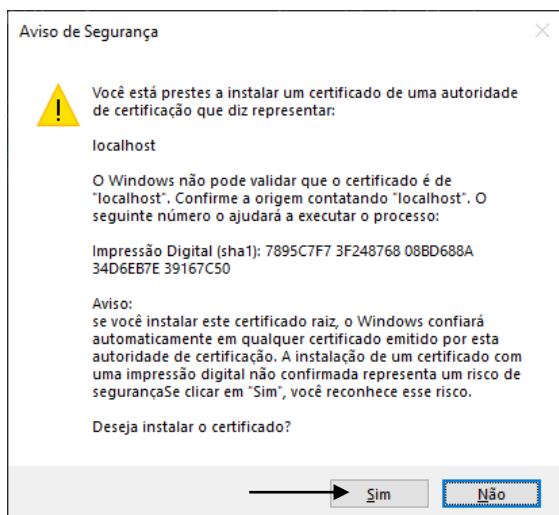
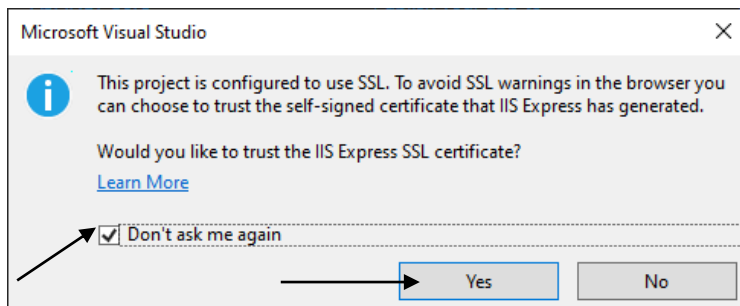
13. Criando um projeto Asp.NET Core no Visual studio 2019

Projeto para cadastro de alunos



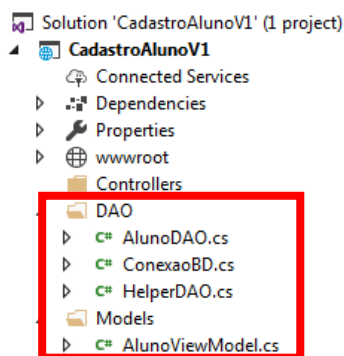


Caso as mensagens abaixo sejam exibidas, proceda com as seguintes escolhas:



Model: Adicionando arquivos para acesso ao banco de dados

Adicione ao seu projeto as classes abaixo. Você pode copiá-las do projeto anterior.

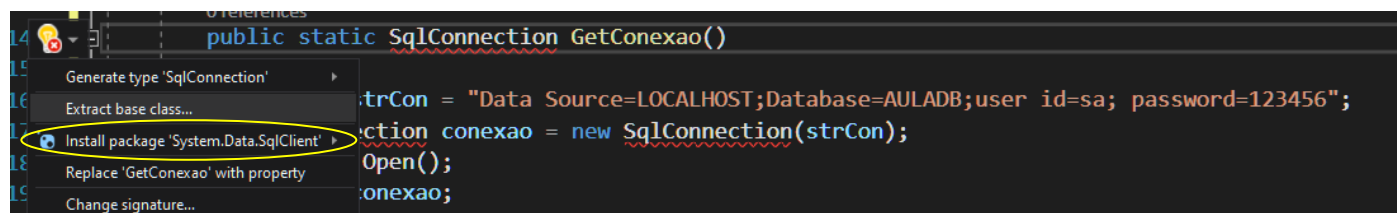


Segue abaixo a listagem completa destas classes:

```
public static class ConexaoBD
{
    public static SqlConnection GetConexao()
    {
        string strCon = "Data Source=LOCALHOST;Initial Catalog=AULADB;user id=sa; password=123456";
        SqlConnection conexao = new SqlConnection(strCon);
        conexao.Open();
        return conexao;
    }
}
```

Obs: Quando estiver utilizando o .NET Core 3.1, será necessário adicionar a biblioteca SqlClient. Para tanto proceda da seguinte forma:

1 – Clique sobre a classe `SqlConnection` e pressione control + ponto (.). Selecione a opção indicada:



```
public class AlunoViewModel
{
    public int Id { get; set; }
    public string Nome { get; set; }
    public double Mensalidade { get; set; }
    public int CidadeId { get; set; }
    public DateTime DataNascimento { get; set; }
}
```

```

public static class HelperDAO
{
    public static void ExecutaSQL(string sql, SqlParameter[] parametros)
    {
        using (SqlConnection conexao = ConexaoBD.GetConexao())
        {
            using (SqlCommand comando = new SqlCommand(sql, conexao))
            {
                if (parametros != null)
                    comando.Parameters.AddRange(parametros);
                comando.ExecuteNonQuery();
            }
            conexao.Close();
        }
    }

    public static DataTable ExecutaSelect(string sql, SqlParameter[] parametros)
    {
        using (SqlConnection conexao = ConexaoBD.GetConexao())
        {
            using (SqlDataAdapter adapter = new SqlDataAdapter(sql, conexao))
            {
                if (parametros != null)
                    adapter.SelectCommand.Parameters.AddRange(parametros);
                DataTable tabela = new DataTable();
                adapter.Fill(tabela);
                conexao.Close();
                return tabela;
            }
        }
    }
}

```

```

public class AlunoDAO
{
    public void Inserir(AlunoViewModel aluno)
    {
        string sql =
            "insert into alunos(id, nome, mensalidade, cidadeId, dataNascimento)" +
            "values ( @id, @nome, @mensalidade, @cidadeId, @dataNascimento)";

        HelperDAO.ExecutaSQL(sql, CriaParametros(aluno));
    }

    public void Alterar(AlunoViewModel aluno)
    {
        string sql =
            "update alunos set nome = @nome, " +
            "mensalidade = @mensalidade, " +
            "cidadeId = @cidadeId, " +
            "dataNascimento = @dataNascimento where id = @id";

        HelperDAO.ExecutaSQL(sql, CriaParametros(aluno));
    }

    private SqlParameter[] CriaParametros(AlunoViewModel aluno)
    {
        SqlParameter[] parametros = new SqlParameter[5];
        parametros[0] = new SqlParameter("id", aluno.Id);
        parametros[1] = new SqlParameter("nome", aluno.Nome);
        parametros[2] = new SqlParameter("mensalidade", aluno.Mensalidade);
        parametros[3] = new SqlParameter("cidadeId", aluno.CidadeId);
        parametros[4] = new SqlParameter("dataNascimento", aluno.DataNascimento);

        return parametros;
    }

    public void Excluir(int id)
    {
        string sql = "delete alunos where id =" + id;
        HelperDAO.ExecutaSQL(sql, null);
    }

    private AlunoViewModel MontaAluno(DataRow registro)
    {
        AlunoViewModel a = new AlunoViewModel();
        a.Id = Convert.ToInt32(registro["id"]);
        a.Nome = registro["nome"].ToString();
        a.CidadeId = Convert.ToInt32(registro["cidadeId"]);
        a.DataNascimento = Convert.ToDateTime(registro["dataNascimento"]);
        a.Mensalidade = Convert.ToDouble(registro["mensalidade"]);
        return a;
    }
}

```

```

public AlunoViewModel Consulta(int id)
{
    string sql = "select * from alunos where id = " + id;
    DataTable tabela = HelperDAO.ExecutaSelect(sql, null);

    if (tabela.Rows.Count == 0)
        return null;
    else
        return MontaAluno(tabela.Rows[0]);
}

public List<AlunoViewModel> Listagem()
{
    List<AlunoViewModel> lista = new List<AlunoViewModel>();

    string sql = "select * from alunos order by nome";
    DataTable tabela = HelperDAO.ExecutaSelect(sql, null);

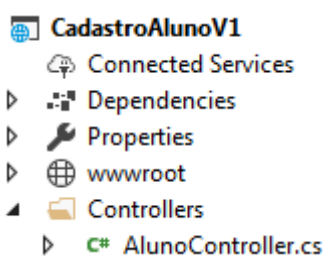
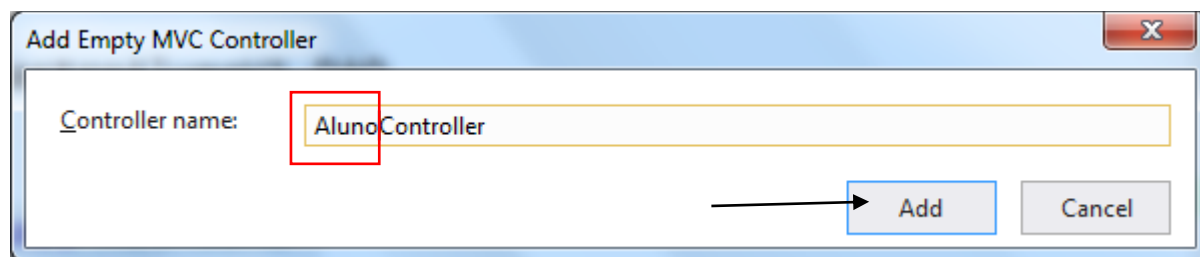
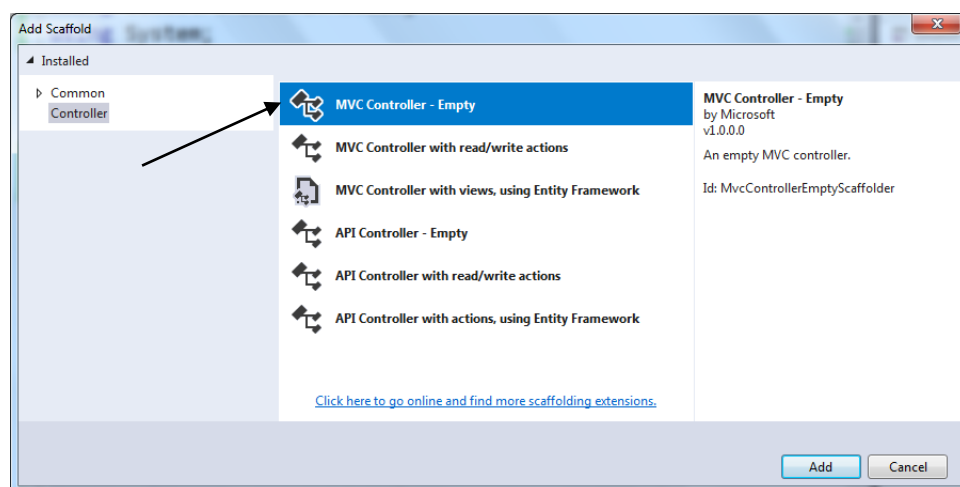
    foreach (DataRow registro in tabela.Rows)
        lista.Add(MontaAluno(registro));
    return lista;
}
}

```

Controller

Na pasta “Controllers” do projeto adicione uma nova classe denominada **AlunoController**:

Botão direito na pasta Controllers -> ADD -> Controller:



Adicione o código destacado no método Index:

```
namespace CadastroAlunoV1.Controllers
{
    public class AlunoController : Controller
    {
        public IActionResult Index()
        {
            AlunoDAO dao = new AlunoDAO();
            List<AlunoViewModel> lista = dao.Listagem();
            return View(lista);
        }
    }
}
```

Esta controller tem neste momento apenas 1 método que será o responsável por listar os dados do aluno em uma página. A página chamada será a Index.cshtml, e por ela ter o mesmo nome do método, não será necessário indicar o seu nome no método View.

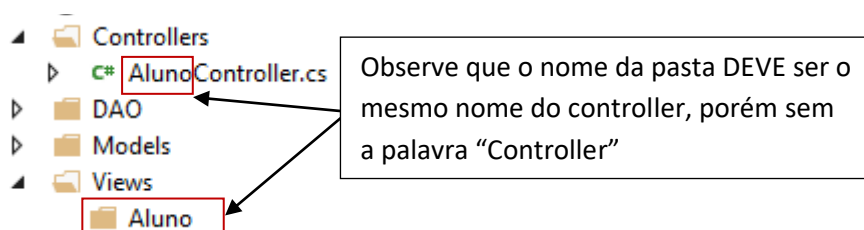
View (arquivos Razor)

Todos os arquivos do Razor terminam com .cshtml. A maioria dos arquivos do Razor deve ser navegável e conter uma mistura de códigos do lado do cliente e do servidor, que, quando processados, fazem com que o HTML seja enviado ao navegador. Essas páginas são geralmente chamadas de "páginas de conteúdo".

<https://www.learnrazorpages.com/razor-pages/files>

Vamos criar uma página para listar todos os registros retornados pelo método Index do controller. O nome da view precisa ser Index. Crie-a como na figura abaixo.

Primeiro, crie dentro da pasta Views uma nova pasta com o nome **Aluno**.



Dentro da pasta Aluno, crie uma nova view com o nome Index.cshtml:

Botão direito na pasta Aluno -> ADD -> VIEW:

Add MVC View

View name:

Template:

Model class:

Options:

☐ Create as a partial view

☐ Reference script libraries

☒ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

Código fonte do Index.cshtml:

```
@model List<AlunoViewModel>
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Listagem de Alunos</h2>

<table class="table table-responsive">
    <tr>
        <th>Ações</th>
        <th>RA</th>
        <th>Nome</th>
        <th>Data de nascimento</th>
    </tr>

    @foreach (var aluno in Model)
    {
        <tr>
            <td></td>
            <td>@aluno.Id</td>
            <td>@aluno.Nome</td>
            <td>@aluno.DataNascimento.ToShortDateString()</td>
        </tr>
    }
</table>
```

Configurando o controller inicial

Edite o arquivo Startup.cs alterando o seguinte trecho no final dele:

► C# Startup.cs

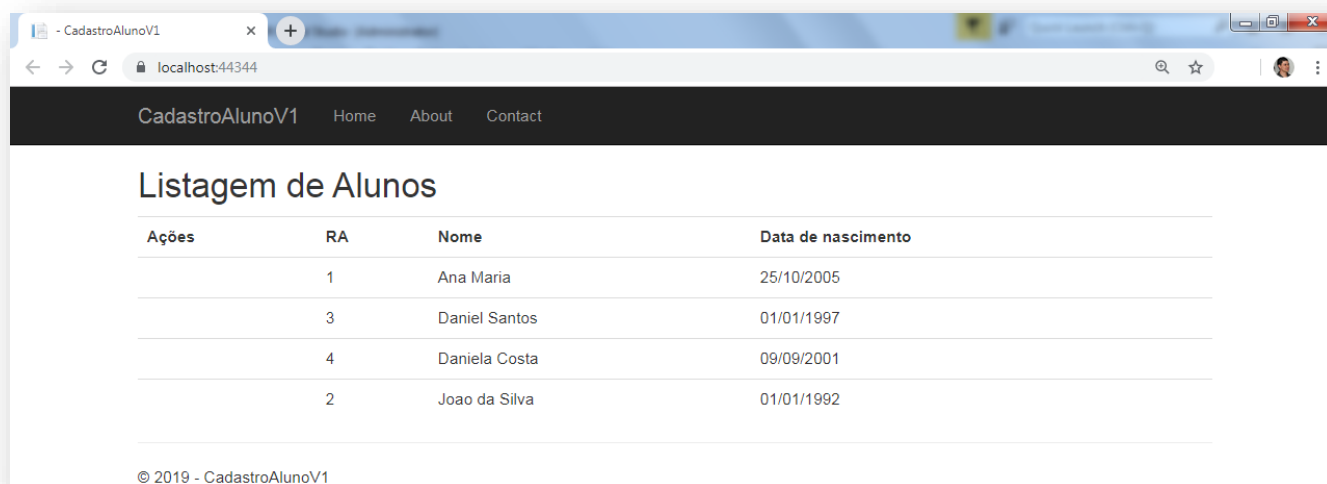
Altere de:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

Para:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Aluno}/{action=Index}/{id?}");
});
```

Executando o site



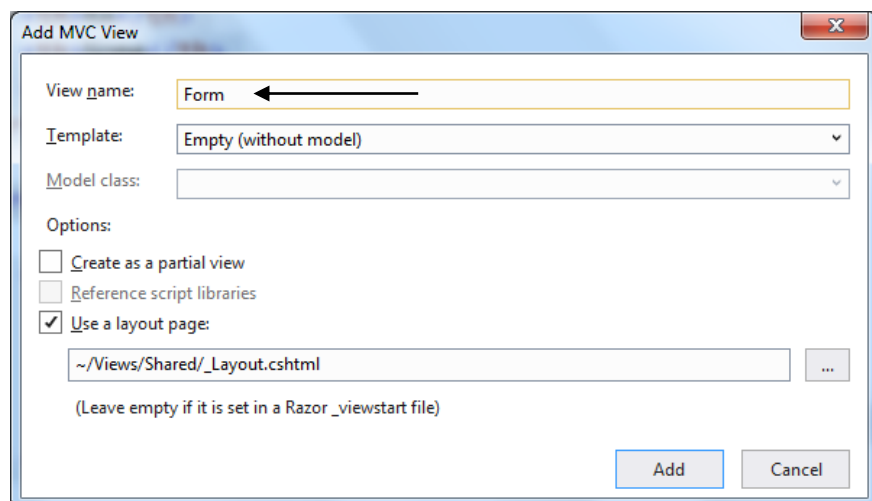
Exercícios:

13.1) Faça um novo website para listar os jogos, assim como fizemos com os alunos.

14. Cadastrando um novo aluno

Iremos criar um formulário que servirá inicialmente para consultar os dados do aluno, permitindo que os mesmos sejam modificados. Posteriormente este mesmo formulário será utilizado para novos cadastros.

Clique com o botão direito sobre a pasta Views -> Aluno -> Add -> View



Código fonte do formulário: Versão simples

```
@model AlunoViewModel
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Cadastro de Aluno</h2>

<form action="/Salvar" method="post">
    <label for="Id" class="control-label">RA</label>
    <input type="number" Name="Id" value="@Model.Id" />
    <br />

    <label for="Nome" class="control-label">Nome do aluno</label>
    <input type="text" Name="Nome" value="@Model.Nome" />
    <br />

    <label for="Mensalidade" class="control-label">Mensalidade</label>
    <input type="text" Name="Mensalidade" value="@Model.Mensalidade" />
    <br />

    <label for="DataNascimento" class="control-label">Data de nascimento</label>
    <input type="date" Name="DataNascimento" value="@Model.DataNascimento.ToString("yyyy-MM-dd")" />
    <br />

    <label for="CidadeId" class="control-label">Código da cidade</label>
    <input type="number" Name="CidadeId" value="@Model.CidadeId" />
    <br />

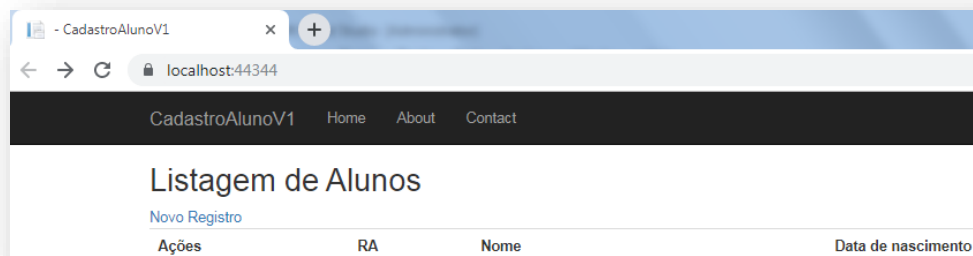
    <input type="submit" value="Salvar dados" />
    <br />
</form>
```

A chamada deste formulário será através de um link na listagem de alunos (index). Adicione o código **destacado** a seguir no index.cshtml:

```
<h2>Listagem de Alunos</h2>
```

```
<a href="/aluno/create">Novo Registro</a>
<br />
```

```
<table class="table table-responsive">
```



Adicione o seguinte método no AlunoController:

```
public IActionResult Create()
{
    AlunoViewModel aluno = new AlunoViewModel();
    aluno.DataNascimento = DateTime.Now;
    return View("Form", aluno);
}

public IActionResult Salvar(AlunoViewModel aluno)
{
    AlunoDAO dao = new AlunoDAO();
    dao.Inserir(aluno);
    return RedirectToAction("index");
}
```

Exercícios:

14.1-) No projeto do cadastro de jogos, faça a opção para cadastrar um novo jogo.

15. Armazenando e recuperando valores nulo (null)

Caso seja necessário armazenar no banco de dados valores nulos, situação comum em campos não obrigatórios, temos que ter um cuidado especial. A primeira coisa que precisamos fazer é configurar nossa variável para aceitar valores nulos. O tipo de dado string aceita naturalmente valores nulos, mas outros tipos não. Para que um tipo de dado aceite valores nulos, precisamos por na frente dele uma interrogação. Vamos fazer alguns ajustes para que seja possível preencher o cadastro sem salvar o valor da mensalidade (este campo já está preparado para aceitar nulo no banco de dados).

Primeiro, ajuste a variável na classe abaixo adicionando a **interrogação**:

```
public class AlunoViewModel
{
    public double? Mensalidade { get; set; }
```

Depois, ajuste a classe **AlunoDAO** para que seja possível armazenar/recuperar valores nulos:

```
private SqlParameter[] CriaParametros(AlunoViewModel aluno)
{
    SqlParameter[] parametros = new SqlParameter[5];
    ...
    if (aluno.Mensalidade == null)
        parametros[2] = new SqlParameter("mensalidade", DBNull.Value);
    else
        parametros[2] = new SqlParameter("mensalidade", aluno.Mensalidade);
    ...
}

private AlunoViewModel MontaAluno(DataRow registro)
{
    AlunoViewModel a = new AlunoViewModel();
    ...
    if (registro["mensalidade"] != DBNull.Value)
        a.Mensalidade = Convert.ToDouble(registro["mensalidade"]);
    ...
}
```

Agora temos que ajustar a listagem de alunos para tratar um possível valor nulo. Altere o código abaixo no **Index.cshtml**

```
@foreach (AlunoViewModel a in Model)
{
    <tr>
        ...
        <td>@a.Mensalidade?.ToString("0.00")</td>
        ...
    </tr>
}
```

16. Exibindo erros

Há diversos tipos de mensagens de erro que podem ocorrer durante a utilização do sistema. Os erros decorrentes de regras de negócio nós veremos mais a frente. Aqui, nós veremos como exibir mensagens decorrentes de controle de exceção. Normalmente esse tipo de erro não é exibido ao usuário, sendo apenas exibida uma mensagem de alerta e um log é então gerado para análise futura. Neste momento do curso, nós iremos exibir os erros em uma tela de erro separada. Isso vai nos ajudar a encontrar os problemas que ocorrem durante o processo de desenvolvimento, porém é importante frisar que este tipo de abordagem não é a mais indicada para o sistema em produção.

O projeto MVC que estamos utilizando já possui uma estrutura básica para exibir os erros em uma tela separada. Iremos fazer pequenos ajustes nesta estrutura. Vamos lá!

ErrorViewModel.cs

Adicione o código abaixo nesta classe:

```
public class ErrorViewModel
{
    public ErrorViewModel(string erro)
    {
        this.Erro = erro;
    }

    public ErrorViewModel()
    {
    }

    public string Erro { get; set; }
    public string RequestId { get; set; }
    public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
}
```

Error.cshtml

Edite \Shared\Error.cshtml alterando o conteúdo para o abaixo:

```
@model ErrorViewModel
@{
    ViewData["Title"] = "Erro";
}

<h2 class="text-danger">Um erro ocorreu durante o processamento da sua requisição. </h2>

<p>
    <strong>Detalhes do erro: </strong> <code> @Model.Erro </code>
</p>
```

Agora, devemos colocar dentro das nossas controllers os blocos try-catch, adicionando no catch o redirecionamento para a página de erro. Ex:

```
public IActionResult Salvar(AlunoViewModel aluno)
{
    try
    {
        AlunoDAO dao = new AlunoDAO();
        dao.Inserir(aluno);
        return RedirectToAction("index");
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}
```

Experimente agora cadastrar dois alunos com o mesmo código.

Exercícios:

16.1) Adicione o controle de erros no projeto de jogos.

17. Enviando parâmetros pela URL para o controller: QueryString

A QueryString é um modelo clássico de manutenção do estado da página. Elas são nada mais do que conjuntos de pares/valores anexados a URL, em diversos sites hoje em dia vemos o uso delas. Seu uso é simples, após a URL de determinada página, adicionamos o primeiro valor usando a seguinte sintaxe: ?Chave=Valor. Para passarmos mais de um conjunto, os mesmos devem ser concatenados usando o caractere coringa &.

Fonte: <https://www.devmedia.com.br/conceitos-e-exemplo-pratico-usando-querystring/18094>

Para enviar parâmetros para o controller através de uma URL, basta seguir este padrão: Na frente da sua action, você deverá colocar o caractere de interrogação (?) seguido de um conjunto de variável e valor.

Exemplo: para passar o id de um registro para uma controller de Funcionarios, você deveria fazer o seguinte:

`/Funcionario/Alterar?id=10`

Neste exemplo, o controler é o **Funcionario**, a action é **Alterar** e a variável que desejamos enviar é o **Id** cujo valor é **10**.

No controller Funcionario, o seu método Alterar deveria ter aproximadamente a seguinte assinatura:

```
Public IActionResult Alterar(id int)
```

Caso queira enviar mais de um parâmetro, você deve separá-los pelo caractere &. EX:

`/Funcionarios/Alterar?id=10&cidade=20`

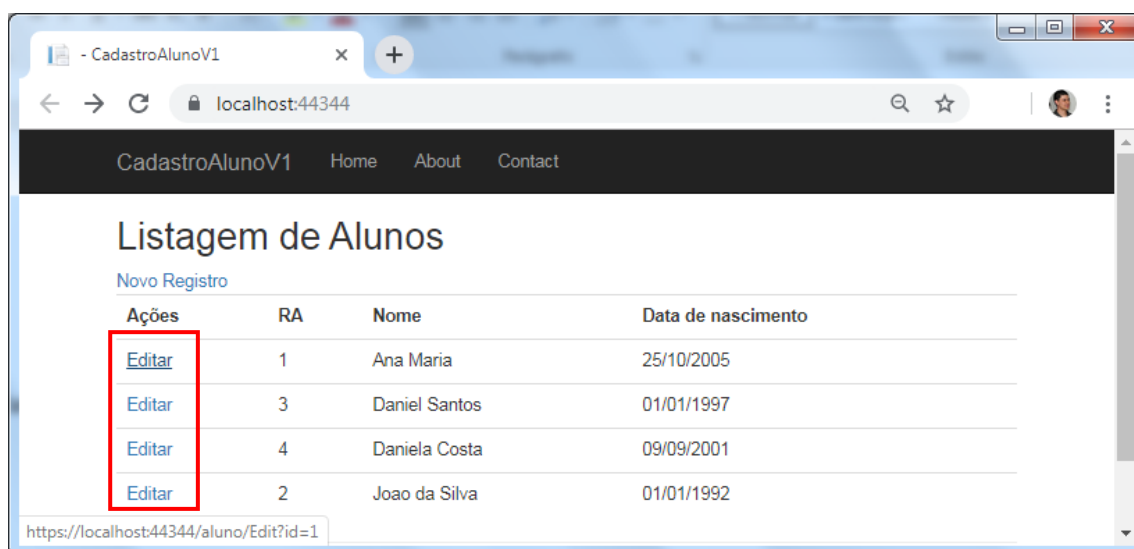
No controller Funcionario, o seu método Alterar deveria ter aproximadamente a seguinte assinatura:

```
Public IActionResult Alterar(id int, int cidade)
```

18. Consultando e alterando os dados do aluno

Vamos adicionar na listagem de alunos (index.cshtml) um link para exibir os dados do aluno. Adicione o código destacado abaixo:

```
@foreach (var aluno in Model)
{
    <tr>
        <td><a href="/aluno/Edit?id=@aluno.Id">Editar</a> </td>
        <td>@aluno.Id</td>
```



Adicione no AlunoController o seguinte método, que será executado quando o usuário clicar no link “Editar”:

```
public IActionResult Edit(int id)
{
    try
    {
        AlunoDAO dao = new AlunoDAO();
        AlunoViewModel aluno = dao.Consulta(id);
        if (aluno == null)
            return RedirectToAction("index");
        else
            return View("Form", aluno);
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}
```

Observe que o formulário é o mesmo utilizado para incluir novos registros. Ao salvar, o método também será o mesmo, sendo assim precisamos fazer uma alteração na Action Salvar para que ele consiga distinguir entre inserir e alterar. Inicialmente utilizaremos uma solução simplificada neste caso que será a de verificar se o registro já existe

no banco de dados, alterando-o. Caso não exista, iremos adicioná-lo. Posteriormente adotaremos uma solução mais sofisticada.

No `AlunoController`, altere o método `Salvar()`, colocando este código:

```
public IActionResult Salvar(AlunoViewModel aluno)
{
    try
    {
        AlunoDAO dao = new AlunoDAO();
        if (dao.Consulta(aluno.Id) == null)
            dao.Inserir(aluno);
        else
            dao.Alterar(aluno);

        return RedirectToAction("index");
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}
```

Exercícios:

18.1) Inclua a opção para alterar um registro no cadastro de jogos. Tente utilizar uma imagem como link para edição.

19. Excluindo registros

Antes de excluir um registro é uma boa prática confirmar a operação junto ao usuário. Esta confirmação será feita utilizando-se a linguagem Javascript. Iremos, a princípio, colocar o javascript dentro da própria página, junto com o Html.

Link para exclusão:

Primeiro, iremos adicionar o **link** para excluir o registro dentro do index.cshtml. Observe que o link irá executar uma função javascript.

```
<tr>
  <td>
    <a href="/aluno/Edit?id=@aluno.Id">Editar</a>
    <a href="javascript:apagarAluno(@aluno.Id)">Apagar</a>
  </td>
```

No final do mesmo arquivo (index.cshtml), logo após o fim da tabela, adicione o seguinte **trecho** de código javascript:

```
</table>

<script>
  function apagarAluno(id)
  {
    if (confirm('Confirma a exclusão do registro?'))
      location.href = 'aluno/Delete?id=' + id;
  }
</script>
```

No controller, adicione o método para executar a exclusão do registro:

```
public IActionResult Delete(int id)
{
    try
    {
        AlunoDAO dao = new AlunoDAO();
        dao.Excluir(id);
        return RedirectToAction("index");
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}
```

20. Sugerindo o próximo código de aluno disponível

O próprio sistema pode indicar o próximo ID de aluno disponível durante uma inclusão. Vamos alterar o programa para que, assim que o usuário clicar no link para criar um novo registro, o sistema automaticamente preencha o campo ID com o próximo valor disponível.

Inclua o seguinte método na classe AlunoDAO:

```

public int ProximoId()
{
    string sql = "select isnull(max(id) +1, 1) as 'MAIOR' from alunos";
    DataTable tabela = HelperDAO.ExecutaSelect(sql, null);
    return Convert.ToInt32(tabela.Rows[0][ "MAIOR" ]);
}

```

Altere o método Create dentro do controller do aluno, adicionando os comandos abaixo:

```

public IActionResult Create()
{
    try
    {
        AlunoViewModel aluno = new AlunoViewModel();
        aluno.DataNascimento = DateTime.Now;

        AlunoDAO dao = new AlunoDAO();
        aluno.Id = dao.ProximoId();

        return View("Form", aluno);
    }
    catch(Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}

```

Exercícios:

20.1) Inclua a opção para excluir um registro no projeto de jogos. Faça também a sugestão de próximo código.

21. ViewBag – Passando dados para a View

Ref. <https://docs.microsoft.com/pt-br/aspnet/core/mvc/views/overview?view=aspnetcore-5.0>

ViewBag é útil para passar pequenas quantidades de dados entre controladores e exibições (Views). Esta abordagem é resolvida dinamicamente em tempo de execução e, portanto, está propensa a causar erros de runtime. Algumas equipes de desenvolvimento a evitam.

Deriva de ViewData de forma que permite a criação de propriedades dinâmicas. Exemplo:
 ViewBag.SomeKey = <value or object>. Nenhuma conversão é necessária. A sintaxe de ViewBag torna mais rápido adicioná-lo a controladores e exibições.

Sendo assim, você pode utilizar a ViewBag para enviar qualquer tipo de informação da controller para a sua view. Após o processamento da view os dados da ViewBag são perdidos.

22. Desabilitando o campo Id nas alterações

Normalmente não permitimos a alteração da chave primária. Sendo assim, iremos bloquear este campo em edições. Este controle também permitirá saber exatamente se estamos inserindo ou editando no momento de salvar a informação. Para fazer este bloqueio, iremos utilizar a ViewBag para enviar dados da Controller para a View.

Primeiro, iremos alterar o controller, alterando os métodos abaixo adicionado as linhas destacadas:

```
public IActionResult Create(int id)
{
    ViewBag.Operacao = "I";
```

```
public IActionResult Edit(int id)
{
    try
    {
        ViewBag.Operacao = "A";
```

No formulário (form.cshtml) de inclusão/alteração, vamos criar um campo escondido para salvar o tipo da operação, além de deixar somente leitura o campo Id no caso de edição. Veja que não utilizamos o atributo “disabled” pois campos de formulário “disabled” não são enviados ao servidor!

```
<div class="row">
    <div class="col-md-4">
        <form action="Salvar" method="post">
            <input type="hidden" name="Operacao" value="@ViewBag.Operacao" />

            <label for="Id" class="control-label">RA</label>
            @if (ViewBag.Operacao == "I")
            {
                <input type="number" Name="Id" value="@Model.Id" class="form-control" />
            }
            else
            {
                <input type="number" Name="Id" value="@Model.Id" class="form-control disabled" readonly/>
            }
            <span asp-validation-for="Id" class="text-danger"></span>
        <br />
```

Altere o método Salvar do Controller para que ele receba do formulário o conteúdo do campo escondido Operacao, para que possamos assim saber se estamos incluindo ou alterando.

```
public IActionResult Salvar(AlunoViewModel aluno, string Operacao)
{
    try
    {
        AlunoDAO dao = new AlunoDAO();
        if (Operacao == "I")
            dao.Inserir(aluno);
        else
            dao.Alterar(aluno);

        return RedirectToAction("index");
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}
```

Exercícios:

22.1) Faça o controle de desabilitar o campo Id durante a edição no projeto de jogos.

23. RAZOR e ASP.NET TAG HELPERS

Fontes:

<https://docs.microsoft.com/pt-br/aspnet/core/mvc/views/razor?view=aspnetcore-3.0>

<https://docs.microsoft.com/pt-br/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-3.0>

<https://docs.microsoft.com/pt-br/aspnet/core/tutorials/first-mvc-app/controller-methods-views?view=aspnetcore-3.1>

Razor é uma sintaxe de marcação para inserir código baseado em servidor em páginas da Web. A sintaxe Razor é composta pela marcação Razor, por C# e por HTML. Arquivos que contêm Razor geralmente têm a extensão de arquivo *.cshtml*. O Razor também é encontrado em arquivos de [Componentes do Razor](#) (*.razor*).

A linguagem padrão do Razor padrão é o HTML. Renderizar HTML da marcação Razor não é diferente de renderizar HTML de um arquivo HTML. A marcação HTML em arquivos Razor *.cshtml* é renderizada pelo servidor inalterado.

```
@{
    var joe = new Person("Joe", 33);
}
```

```
<p>Age @(joe.Age)</p>
```

O que são Auxiliares de Marca (TAG HELPERS)

Os Auxiliares de Marca permitem que o código do lado do servidor participe da criação e renderização de elementos HTML em arquivos do Razor. Por exemplo, o ImageTagHelper interno pode acrescentar um número de versão ao nome da imagem. Sempre que a imagem é alterada, o servidor gera uma nova versão exclusiva para a imagem, de modo que os clientes tenham a garantia de obter a imagem atual (em vez de uma imagem obsoleta armazenada em cache). Há muitos Auxiliares de Marca internos para tarefas comuns – como criação de formulários, links, carregamento de ativos e muito mais – e ainda outros disponíveis em repositórios GitHub públicos e como NuGet. Os Auxiliares de Marca são criados no C# e são direcionados a elementos HTML de acordo com o nome do elemento, o nome do atributo ou a marca pai. Por exemplo, o LabelTagHelper interno pode ser direcionado ao elemento <label> HTML quando os atributos LabelTagHelper são aplicados. Se você está familiarizado com [Auxiliares HTML](#), os Auxiliares de Marca reduzem as transições explícitas entre HTML e C# em exibições do Razor. Em muitos casos, os Auxiliares HTML fornecem uma abordagem alternativa a um Auxiliar de Marca específico, mas é importante reconhecer que os Auxiliares de Marca não substituem os Auxiliares HTML e que não há um Auxiliar de Marca para cada Auxiliar HTML. [Comparação entre Auxiliares de Marca e Auxiliares HTML](#) explica as diferenças mais detalhadamente.

```
<input asp-for="Nome" class="form-control" />
```

No lugar de:

```
<input type="text" Name="Nome" Id="Nome" value="@Model.Nome" class="form-control" />
```

Outros exemplos:

```
<a asp-action="Edit" asp-route-id="@item.ID">Edit</a>
```

```
<form asp-action="Edit">
```

24. Validando os campos do formulário

A validação de dados deve ocorrer, de forma incondicional, no controller, já que validação no formulário e com javascript pode ser burlada.

Crie o método para validar os dados dentro do controller

```
private void ValidaDados(AlunoViewModel aluno, string operacao)
{
    ModelState.Clear(); // limpa os erros criados automaticamente pelo Asp.net (que podem estar com msg em inglês)
    AlunoDAO dao = new AlunoDAO();
    if (operacao == "I" && dao.Consulta(aluno.Id) != null)
        ModelState.AddModelError("Id", "Código já está em uso.");
    if (operacao == "A" && dao.Consulta(aluno.Id) == null)
        ModelState.AddModelError("Id", "Aluno não existe.");
    if (aluno.Id <= 0)
        ModelState.AddModelError("Id", "Id inválido!");

    if (string.IsNullOrEmpty(aluno.Nome))
        ModelState.AddModelError("Nome", "Preencha o nome.");

    if (aluno.Mensalidade < 0)
        ModelState.AddModelError("Mensalidade", "Campo obrigatório.");
    if (aluno.CidadeId <= 0)
        ModelState.AddModelError("CidadeId", "Informe o código da cidade.");

    if (aluno.DataNascimento > DateTime.Now)
        ModelState.AddModelError("DataNascimento", "Data inválida!");
}
```

O método salvar vai ficar da seguinte forma:

```
public IActionResult Salvar(AlunoViewModel aluno, string Operacao)
{
    try
    {
        ValidaDados(aluno, Operacao);
        if (ModelState.IsValid == false)
        {
            ViewBag.Operacao = Operacao;
            return View("Form", aluno);
        }
        else
        {
            AlunoDAO dao = new AlunoDAO();
            if (Operacao == "I")
                dao.Inserir(aluno);
            else
                dao.Alterar(aluno);

            return RedirectToAction("index");
        }
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}
```

É muito importante preencher corretamente a operação ao retornar à tela!

O código fonte do formulário deverá incluir as tags responsáveis para exibir os erros.

Código fonte do formulário: Versão já com os campos de Validação, tag helpers e um link para retornar à listagem:

```
@model AlunoViewModel
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Cadastro de Aluno</h2>

<div class="row">
    <div class="col-md-4">
        <form action="Salvar" method="post">
            <input type="hidden" name="Operacao" value="@ViewBag.Operacao" />

            <label for="Id" class="control-label">RA</label>
            @if (ViewBag.Operacao == "I")
            {
                <input asp-for="Id" class="form-control" />
            }
            else
            {
                <input asp-for="Id" class="form-control" readonly />
            }
            <span asp-validation-for="Id" class="text-danger"></span>

            <br />

            <label for="Nome" class="control-label">Nome do aluno</label>
            <input asp-for="Nome" class="form-control" />
            <span asp-validation-for="Nome" class="text-danger"></span>
            <br />

            <label for="Mensalidade" class="control-label">Mensalidade</label>
            <input asp-for="Mensalidade" class="form-control" />
            <span asp-validation-for="Mensalidade" class="text-danger"></span>
            <br />

            <label for="DataNascimento" class="control-label">Data de nascimento</label>
            <input type="date" Name="DataNascimento" value="@Model.DataNascimento.ToString("yyyy-MM-dd")" class="form-control" />
            <span asp-validation-for="DataNascimento" class="text-danger"></span>
            <br />

            <label for="CidadeId" class="control-label">Código da cidade</label>
            <input asp-for="CidadeId" class="form-control" />
            <span asp-validation-for="CidadeId" class="text-danger"></span>
            <br />

            <input type="submit" value="Salvar dados" />
            &nbsp;
            <a href="/aluno/index" class="btn btn-light">Voltar</a>

            <br />
        </form>
    </div>
</div>
```

Cadastro de Aluno

RA

3

Código já está em uso.

Nome do aluno

Preencha o nome.

Mensalidade

-1

Campo obrigatório.

Data de nascimento

03/10/2019

Data inválida!

Código da cidade

0

Informe o código da cidade.

Salvar dados

Exercícios:

24.1-) Faça as validações no cadastro de jogos. Todos os campos são obrigatórios.

24.2-) Crie um novo projeto para fazer o cadastro de funcionários. Adicione as opções de listar, incluir, alterar e excluir dados. Faça as validações, sendo que todos os campos são obrigatórios. Os tipos válidos de funcionário são “V” para vendedores, “G” para gerentes, “E” engenheiros e “O” para outros. Tente utilizar uma caixa combo para o campo tipo. O campo ativo deve ser um checkbox. Na listagem dos dados, tente exibir na coluna tipo a descrição do tipo. Tente utilizar o SweetAlert para exibir a mensagem de confirmação de exclusão

(<https://lipis.github.io/bootstrap-sweetalert/>)

Tabela:

Create table Funcionarios (func_id int primary key, func_nome varchar(max), func_tipo char(1), func_ativo bit)

25. Alterações no CRUD para realizar as operações utilizando Stored Procedures

Stored procedures utilizadas neste exemplo:

```
create procedure spIncluiAluno
(
    @id int,
    @nome varchar(max),
    @mensalidade money,
    @cidadeId int,
    @DataNascimento datetime
)
as
begin
    insert into Alunos
    (id, nome, mensalidade, cidadeId, DataNascimento)
    values
    (@id, @nome, @mensalidade, @cidadeId, @DataNascimento)
end
GO

create procedure spAlterarAluno
(
    @id int,
    @nome varchar(max),
    @mensalidade money,
    @cidadeId int,
    @DataNascimento datetime
)
as
begin
    update alunos set
        nome = @nome,
        mensalidade = @mensalidade,
        cidadeId = @cidadeId,
        dataNascimento = @dataNascimento
    where id = @id
end
GO

create procedure spExcluiAluno
(
    @id int
)
as
begin
    delete alunos where id = @id
end
GO

create procedure spConsultaAluno
(
    @id int
)
as
begin
    select * from Alunos where id = @id
end
GO

create procedure spListagemAlunos
as
begin
    select * from Alunos
end
GO

create procedure spProximoId
(@tabela varchar(max))
as
begin
    exec('select isnull(max(id) +1, 1) as MAIOR from '
    +@tabela)
end
GO
```

Crie dois novos métodos na classe **HelperDAO** para permitir a execução de Stored procedures.

```
public static DataTable ExecutaProcSelect(string nomeProc, SqlParameter[] parametros)
{
    using (SqlConnection conexao = ConexaoBD.GetConexao())
    {
        using (SqlDataAdapter adapter = new SqlDataAdapter(nomeProc, conexao))
        {
            if (parametros != null)
                adapter.SelectCommand.Parameters.AddRange(parametros);

            adapter.SelectCommand.CommandType = CommandType.StoredProcedure;

            DataTable tabela = new DataTable();
            adapter.Fill(tabela);
            conexao.Close();
            return tabela;
        }
    }
}
```



```

public static void ExecutaProc(string nomeProc, SqlParameter[] parametros)
{
    using (SqlConnection conexao = ConexaoBD.GetConexao())
    {
        using (SqlCommand comando = new SqlCommand(nomeProc, conexao))
        {
            comando.CommandType = CommandType.StoredProcedure;
            if (parametros != null)
                comando.Parameters.AddRange(parametros);
            comando.ExecuteNonQuery();
        }
        conexao.Close();
    }
}

```

Alterações na classe AlunoDAO:

```

public class AlunoDAO
{
    public void Inserir(AlunoViewModel aluno)
    {
        HelperDAO.ExecutaProc("spIncluiAluno", CriaParametros(aluno));
    }

    public void Alterar(AlunoViewModel aluno)
    {
        HelperDAO.ExecutaProc("spAlterarAluno", CriaParametros(aluno));
    }

    private SqlParameter[] CriaParametros(AlunoViewModel aluno)
    {
        SqlParameter[] parametros = new SqlParameter[5];
        parametros[0] = new SqlParameter("id", aluno.Id);
        parametros[1] = new SqlParameter("nome", aluno.Nome);
        parametros[2] = new SqlParameter("mensalidade", aluno.Mensalidade);
        parametros[3] = new SqlParameter("cidadeId", aluno.CidadeId);
        parametros[4] = new SqlParameter("dataNascimento", aluno.DataNascimento);

        return parametros;
    }

    public void Excluir(int id)
    {
        var p = new SqlParameter[]
        {
            new SqlParameter("id", id)
        };

        HelperDAO.ExecutaProc("spExcluiAluno", p);
    }

    private AlunoViewModel MontaAluno(DataRow registro)
    {
        AlunoViewModel a = new AlunoViewModel();
        a.Id = Convert.ToInt32(registro["id"]);
        a.Nome = registro["nome"].ToString();
        a.CidadeId = Convert.ToInt32(registro["cidadeId"]);
        a.DataNascimento = Convert.ToDateTime(registro["dataNascimento"]);
        a.Mensalidade = Convert.ToDouble(registro["mensalidade"]);
        return a;
    }
}

```

```

public AlunoViewModel Consulta(int id)
{
    var p = new SqlParameter[]
    {
        new SqlParameter("id", id)
    };

    DataTable tabela = HelperDAO.ExecutaProcSelect("spConsultaAluno", p );

    if (tabela.Rows.Count == 0)
        return null;
    else
        return MontaAluno(tabela.Rows[0]);
}

public List<AlunoViewModel> Listagem()
{
    List<AlunoViewModel> lista = new List<AlunoViewModel>();

    DataTable tabela = HelperDAO.ExecutaProcSelect("spListagemAlunos", null);

    foreach (DataRow registro in tabela.Rows)
        lista.Add(MontaAluno(registro));
    return lista;
}

public int ProximoId()
{
    var p = new SqlParameter[]
    {
        new SqlParameter("tabela", "alunos")
    };

    DataTable tabela = HelperDAO.ExecutaProcSelect("spProximoId", p);
    return Convert.ToInt32(tabela.Rows[0][ "MAIOR" ]);
}
}

```

Exercícios

25.1) Crie um CRUD de ferramentas. Utilize stored procedures. Durante a edição desabilite o campo Id e durante a inclusão não exiba este campo.

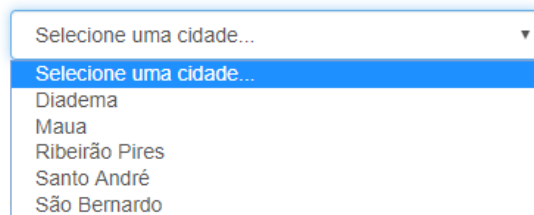
```

CREATE TABLE [dbo].[ferramentas](
    Id [int] NOT NULL primary key identity (1,1),
    descricao [varchar](50) NULL,
    FabricanteId int );

```

26. Listando valores de uma tabela em uma caixa Combo

Cidade



Para exemplificar, vamos utilizar o cadastro criado no tópico anterior, listando as cidades em uma caixa combo. Primeiro, vamos criar os objetos de banco de dados:

```
CREATE TABLE cidades (
    id int NOT NULL primary key,
    nome varchar(30) NULL,
)

GO

--Insira os seguintes registros:

insert into cidades (id,nome) values (1, 'São Bernardo')
insert into cidades (id,nome) values (2, 'Santo André')
insert into cidades (id,nome) values (3, 'Maua')
insert into cidades (id,nome) values (4, 'Diadema')
insert into cidades (id,nome) values (5, 'Ribeirão Pires')
GO

create procedure spListagemCidades
as
begin
    select * from cidades order by nome
end
GO
```

Vamos criar uma model para esta tabela:

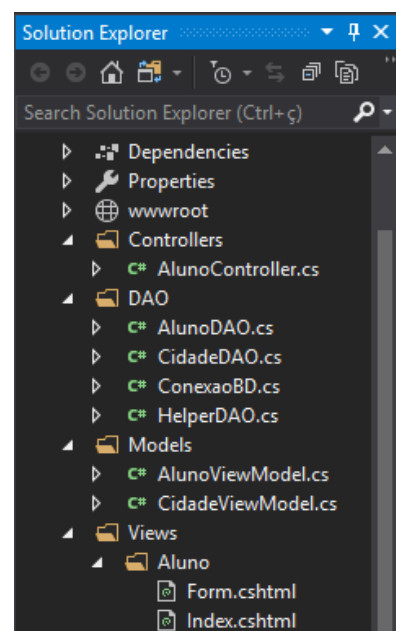
```
public class CidadeViewModel
{
    public int Id { get; set; }
    public string Nome { get; set; }
}
```

Dentro da pasta DAO, crie a classe CidadeDAO:

```
public class CidadeDAO
{
    public List<CidadeViewModel> ListaCidades()
    {
        List<CidadeViewModel> lista = new List<CidadeViewModel>();

        DataTable tabela = HelperDAO.ExecutaProcSelect("spListagemCidades", null);

        foreach (DataRow registro in tabela.Rows)
            lista.Add(MontaCidade(registro));
        return lista;
    }
}
```



```

private CidadeViewModel MontaCidade(DataRow registro)
{
    CidadeViewModel c = new CidadeViewModel()
    {
        Id = Convert.ToInt32(registro["id"]),
        Nome = registro["nome"].ToString()
    };

    return c;
}
}

```

Alterações necessárias no **AlunoController**:

Crie o método abaixo:

```

private void PreparaListaCidadesParaCombo()
{
    CidadeDAO cidadeDao = new CidadeDAO();
    var cidades = cidadeDao.ListaCidades();
    List<SelectListItem> listaCidades = new List<SelectListItem>();

    listaCidades.Add(new SelectListItem("Selecione uma cidade...", "0"));

    foreach (var cidade in cidades)
    {
        SelectListItem item = new SelectListItem(cidade.Nome, cidade.Id.ToString());
        listaCidades.Add(item);
    }
    ViewBag.Cidades = listaCidades;
}

```

Altere os métodos abaixo, adicionando as linhas **destacadas**:

```

public IActionResult Create()
{
    try
    {
        ViewBag.Operacao = "I";
        AlunoViewModel aluno = new AlunoViewModel();
        aluno.DataNascimento = DateTime.Now;

        AlunoDAO dao = new AlunoDAO();
        aluno.Id = dao.ProximoId();

        PreparaListaCidadesParaCombo();
        return View("Form", aluno);
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}

public IActionResult Edit(int id)
{
    try
    {
        ViewBag.Operacao = "A";
        AlunoDAO dao = new AlunoDAO();
        AlunoViewModel aluno = dao.Consulta(id);
        PreparaListaCidadesParaCombo();

        if (aluno == null)
            return RedirectToAction("index");
        else
            return View("Form", aluno);
    }
}

```

```

        catch(Exception erro)
        {
            return View("Error", new ErrorViewModel(erro.ToString()));
        }
    }

    public IActionResult Salvar(AlunoViewModel aluno, string Operacao)
    {
        try
        {
            ValidaDados(aluno, Operacao);
            if (ModelState.IsValid == false)
            {
                ViewBag.Operacao = Operacao;
                PreparaListaCidadesParaCombo();
                return View("Form", aluno);
            }
            else
            {
                AlunoDAO dao = new AlunoDAO();
                if (Operacao == "I")
                    dao.Inserir(aluno);
                else
                    dao.Alterar(aluno);
                return RedirectToAction("index");
            }
        }
        catch(Exception erro)
        {
            return View("Error", new ErrorViewModel(erro.ToString()));
        }
    }
}

```

Alterações no Formulário (Form.cshtml)

```

<label for="CidadeId" class="control-label">Cidade</label>
<select asp-for="CidadeId" asp-items="ViewBag.Cidades" class="form-control"> </select>
<span asp-validation-for="CidadeId" class="text-danger"></span>
<br />

```

Exercícios

26.1) Faça ajustes na última versão do cadastro de jogos para que a classe DAO realize todos os acessos ao banco de dados através de Stored procedures. Adicione também a opção de listar as categorias através de uma caixa combo.

```

Create table Categorias (id int primary key, descricao varchar(max) );
insert into Categorias values (1, 'Ação');
insert into Categorias values (2, 'RPG');
insert into Categorias values (3, 'Corrida');
insert into Categorias values (4, 'Aventura');
insert into Categorias values (5, 'Tiro');

```

27. Herança aplicada aos Models e DAOs

Iremos a seguir fazer alterações no cadastro de Alunos utilizando conceitos de POO para redução de código.

Pressupostos:

1. A chave primária das tabelas deve ser um campo inteiro e ser nomeada como **id**
2. O nome da stored procedure de insert deve ser spInsert_**[NomeDaTabela]** onde nome da tabela é o.... nome da tabela 😊
3. O nome da stored procedure de update deve ser spUpdate_**[NomeDaTabela]** seguindo a mesma regra.

Classe Ancestral para os todas as Models

Dado que toda tabela terá um campo inteiro como chave primária, iremos defini-lo aqui. Isso irá garantir que alguns algoritmos possam ser generalizados nas classes descendentes.

```
public abstract class PadraoViewModel
{
    public virtual int Id { get; set; }
}
```

Stored Procedures Genéricas

A fim de reduzir a quantidade de stored procedures necessárias, algumas foram criadas passando-se por parâmetro o nome da tabela. A ideia é que, em um CRUD normal, apenas as procedures de insert e update devam ser criadas.

```
create procedure spDelete
(
    @id int ,
    @tabela varchar(max)
)
as
begin
    declare @sql varchar(max);
    set @sql = 'delete ' + @tabela +
        ' where id = ' + cast(@id as varchar(max))
    exec(@sql)
end
GO

create procedure spConsulta
(
    @id int ,
    @tabela varchar(max)
)
as
begin
    declare @sql varchar(max);
    set @sql = 'select * from ' + @tabela +
        ' where id = ' + cast(@id as varchar(max))
    exec(@sql)
end
GO

create procedure spDelete
GO

create procedure spListagem
(
    @tabela varchar(max),
    @ordem varchar(max))
as
begin
    exec('select * from ' + @tabela +
        ' order by ' + @ordem)
end
GO

create procedure spProximoId
(@tabela varchar(max))
as
begin
    exec('select isnull(max(id) +1, 1) as MAIOR from '
        +@tabela)
end
GO
```

Stored procedures exclusivas para o cadastro de alunos

```
create procedure spInsert_Alunos
(
    @id int,
    @nome varchar(max),
    @mensalidade money,
    @cidadeId int,
    @DataNascimento datetime
)
as
begin
    insert into Alunos
        (id, nome, mensalidade, cidadeId, DataNascimento)
    values
        (@id, @nome, @mensalidade, @cidadeId, @DataNascimento)
end
GO
```

```
create procedure spUpdate_Alunos
(
    @id int,
    @nome varchar(max),
    @mensalidade money,
    @cidadeId int,
    @DataNascimento datetime
)
as
begin
    update alunos set
        nome = @nome,
        mensalidade = @mensalidade,
        cidadeId = @cidadeId,
        dataNascimento = @dataNascimento
    where id = @id
end
GO
```

Classe DAO Padrão

A classe DAO a seguir será ancestral para todas as classes DAO do sistema, gerado automaticamente as principais instruções para acesso ao banco de dados. A fim de evitar operações de CAST, iremos utilizar a tecnologia Generics do C# que permite definir o tipo de dado da Model a ser utilizada.

```
using CadAlunosMVC_v1.Models;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Threading.Tasks;

namespace CadAlunosMVC_v1.DAO
{
    public abstract class PadraoDAO<T> where T : PadraoViewModel
    {
        public PadraoDAO()
        {
            SetTabela();
        }
    }
}
```

```

protected string Tabela { get; set; }
protected string NomeSpListagem { get; set; } = "spListagem";
protected abstract SqlParameter[] CriaParametros(T model);
protected abstract T MontaModel(DataRow registro);
protected abstract void SetTabela();

public virtual void Insert(T model)
{
    HelperDAO.ExecutaProc("spInsert_" + Tabela, CriaParametros(model));
}

public virtual void Update(T model)
{
    HelperDAO.ExecutaProc("spUpdate_" + Tabela, CriaParametros(model));
}

public virtual void Delete(int id)
{
    var p = new SqlParameter[]
    {
        new SqlParameter("id", id),
        new SqlParameter("tabela", Tabela)
    };
    HelperDAO.ExecutaProc("spDelete", p);
}

public virtual T Consulta(int id)
{
    var p = new SqlParameter[]
    {
        new SqlParameter("id", id),
        new SqlParameter("tabela", Tabela)
    };
    var tabela = HelperDAO.ExecutaProcSelect("spConsulta", p);
    if (tabela.Rows.Count == 0)
        return null;
    else
        return MontaModel(tabela.Rows[0]);
}

public virtual int ProximoId()
{
    var p = new SqlParameter[]
    {
        new SqlParameter("tabela", Tabela)
    };
    var tabela = HelperDAO.ExecutaProcSelect("spProximoId", p);
    return Convert.ToInt32(tabela.Rows[0][0]);
}

public virtual List<T> Listagem()
{
    var p = new SqlParameter[]
    {
        new SqlParameter("tabela", Tabela),
        new SqlParameter("Ordem", "1") // 1 é o primeiro campo da tabela
    };
    var tabela = HelperDAO.ExecutaProcSelect(NomeSpListagem, p);
    List<T> lista = new List<T>();
    foreach (DataRow registro in tabela.Rows)
        lista.Add(MontaModel(registro));
}

```



```

        return lista;
    }
}

```

Alterações no cadastro de alunos para se adequar ao novo padrão.

Agora, iremos fazer alguns ajustes no cadastro de alunos que já havíamos desenvolvido para que ele se adeque ao novo padrão que criamos para a Model e a DAO.

Observe que na classe abaixo, o campo Id foi removido pois agora ele está disponível através da herança.

```

public class AlunoViewModel : PadraoViewModel
{
    public string Nome { get; set; }
    public double Mensalidade { get; set; }
    public int CidadeId { get; set; }
    public DateTime DataNascimento { get; set; }
}

public class AlunoDAO : PadraoDAO<AlunoViewModel>
{
    protected override SqlParameter[] CriaParametros(AlunoViewModel model)
    {
        SqlParameter[] parametros = new SqlParameter[5];
        parametros[0] = new SqlParameter("id", model.Id);
        parametros[1] = new SqlParameter("nome", model.Nome);
        parametros[2] = new SqlParameter("mensalidade", model.Mensalidade);
        parametros[3] = new SqlParameter("cidadeId", model.CidadeId);
        parametros[4] = new SqlParameter("dataNascimento", model.DataNascimento);

        return parametros;
    }

    protected override AlunoViewModel MontaModel(DataRow registro)
    {
        AlunoViewModel a = new AlunoViewModel();
        a.Id = Convert.ToInt32(registro["id"]);
        a.Nome = registro["nome"].ToString();
        a.CidadeId = Convert.ToInt32(registro["cidadeId"]);
        a.DataNascimento = Convert.ToDateTime(registro["dataNascimento"]);
        a.Mensalidade = Convert.ToDouble(registro["mensalidade"]);
        return a;
    }

    protected override void SetTabela()
    {
        Tabela = "alunos";
    }
}

```

Exercícios

27.1) Aplique estes padrões ao cadastro de jogos.

28. Controller Padrão

O objetivo do código a seguir é criar um controlador padrão que servirá de base para os demais, generalizando e reduzindo o código necessário para efetuar as operações básicas de um CRUD.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using CadastroAlunoV1.DAO;
using CadastroAlunoV1.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace CadastroAlunoV1.Controllers
{
    public class PadraoController<T> : Controller where T : PadraoViewModel
    {
        protected PadraoDAO<T> DAO { get; set; }
        protected bool GeraProximoId { get; set; }
        protected string NomeViewIndex { get; set; } = "index";
        protected string NomeViewForm { get; set; } = "form";

        public virtual IActionResult Index()
        {
            try
            {
                var lista = DAO.Listagem();
                return View(NomeViewIndex, lista);
            }
            catch (Exception erro)
            {
                return View("Error", new ErrorViewModel(erro.ToString()));
            }
        }

        public virtual IActionResult Create()
        {
            try
            {
                ViewBag.Operacao = "I";
                T model = Activator.CreateInstance(typeof(T)) as T;
                PreencheDadosParaView("I", model);
                return View(NomeViewForm, model);
            }
            catch (Exception erro)
            {
                return View("Error", new ErrorViewModel(erro.ToString()));
            }
        }

        protected virtual void PreencheDadosParaView(string Operacao, T model)
        {
            if (GeraProximoId && Operacao == "I")
            {
                model.Id = DAO.GeraProximoId();
            }
        }
    }
}
```

```

        model.Id = DAO.ProximoId();
    }

    public virtual IActionResult Save(T model, string Operacao)
    {
        try
        {
            ValidaDados(model, Operacao);
            if (ModelState.IsValid == false)
            {
                ViewBag.Operacao = Operacao;
                PreencheDadosParaView(Operacao, model);
                return View(NomeViewForm, model);
            }
            else
            {
                if (Operacao == "I")
                    DAO.Insert(model);
                else
                    DAO.Update(model);
                return RedirectToAction(NomeViewIndex);
            }
        }
        catch (Exception erro)
        {
            return View("Error", new ErrorViewModel(erro.ToString()));
        }
    }

    protected virtual void ValidaDados(T model, string operacao)
    {
        ModelState.Clear();

        if (operacao == "I" && DAO.Consulta(model.Id) != null)
            ModelState.AddModelError("Id", "Código já está em uso!");
        if (operacao == "A" && DAO.Consulta(model.Id) == null)
            ModelState.AddModelError("Id", "Este registro não existe!");
        if (model.Id <= 0)
            ModelState.AddModelError("Id", "Id inválido!");
    }

    public IActionResult Edit(int id)
    {
        try
        {
            ViewBag.Operacao = "A";
            var model = DAO.Consulta(id);
            if (model == null)
                return RedirectToAction(NomeViewIndex);
            else
            {
                PreencheDadosParaView("A", model);
                return View(NomeViewForm, model);
            }
        }
        catch (Exception erro)
        {
            return View("Error", new ErrorViewModel(erro.ToString()));
        }
    }

    public IActionResult Delete(int id)
    {
        try
        {
            DAO.Delete(id);
        }
    }

```

```

        return RedirectToAction(NomeViewIndex);
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}
}
}

```

Aplicando o controlador Padrão em um CRUD de cidade

Primeiro, vamos criar as stored procedures específicas para o cadastro de cidades.

```

create procedure spInsert_Cidades
(
    @id int,
    @nome varchar(max)
)
as
begin
    insert into Cidades (id, nome) values (@id, @nome)
end
GO

create procedure spUpdate_Cidades
(
    @id int,
    @nome varchar(max)
)
as
begin
    update cidades set nome = @nome where id = @id
end
GO

```

Model

```

public class CidadeViewModel : PadraoViewModel
{
    public string Nome { get; set; }
}

```

DAO

```

public class CidadeDAO : PadraoDAO<CidadeViewModel>
{
    protected override SqlParameter[] CriaParametros(CidadeViewModel model)
    {
        SqlParameter[] parametros =
        {
            new SqlParameter("id", model.Id),
            new SqlParameter("nome", model.Nome)
        };

        return parametros;
    }

    protected override CidadeViewModel MontaModel(DataRow registro)
    {
        CidadeViewModel c = new CidadeViewModel()
        {
            Id = Convert.ToInt32(registro["id"]),
            Nome = registro["nome"].ToString()
        };
    }
}

```

```

        return c;
    }

    protected override void SetTabela()
    {
        Tabela = "Cidades";
    }
}

```

Controller

```

public class CidadeController : PadraoController<CidadeViewModel>
{
    public CidadeController()
    {
        DAO = new CidadeDAO();
        GeraProximoId = true;
    }
    protected override void ValidaDados(CidadeViewModel model, string operacao)
    {
        base.ValidaDados(model, operacao);
        if (string.IsNullOrEmpty(model.Nome))
            ModelState.AddModelError("Nome", "Preencha o nome.");
    }
}

```

Cidade\Index.cshtml

Crie uma pasta chamada Cidade para colocar os arquivos cshtml a seguir.

```

@model List<CidadeViewModel>
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Listagem de Cidades</h2>

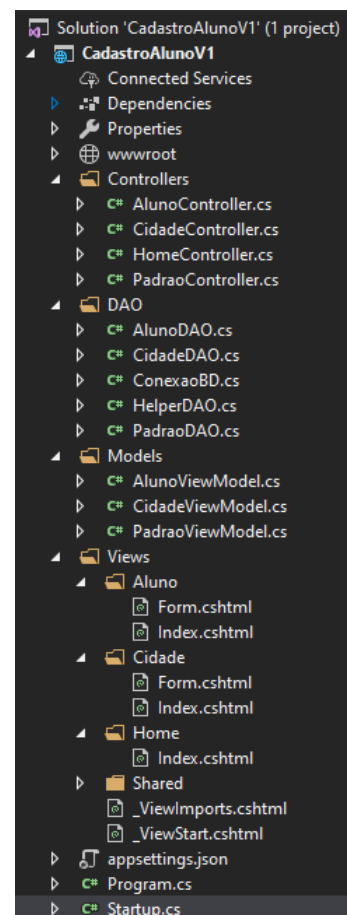
<a href="/cidade/create">Novo Registro</a>
<br />

<table class="table table-responsive">
    <tr>
        <th>Ações</th>
        <th>Código</th>
        <th>Nome</th>
    </tr>

    @foreach (var cidade in Model)
    {
        <tr>
            <td>
                <a href="/cidade/Edit?id=@cidade.Id">Editar</a>
                <a href="javascript:apagarCidade(@cidade.Id)">Apagar</a>
            </td>
            <td>@cidade.Id</td>
            <td>@cidade.Nome</td>
        </tr>
    }
</table>

<script>

```



```
function apagarCidade(id)
{
    if (confirm('Confirma a exclusão do registro?'))
        location.href = 'cidade/Delete?id=' + id;
}
</script>
```

Cidade\Form.cshtml

```
@model CidadeViewModel
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Cadastro de Cidade</h2>

<div class="row">
    <div class="col-md-4">
        <form asp-action="Salvar">
            <input type="hidden" name="Operacao" value="@ViewBag.Operacao" />

            <label for="Id" class="control-label">RA</label>
            @if (ViewBag.Operacao == "I")
            {
                <input asp-for="Id" class="form-control" />
            }
            else
            {
                <input asp-for="Id" class="form-control" readonly />
            }
            <span asp-validation-for="Id" class="text-danger"></span>
            <br />

            <label for="Nome" class="control-label">Nome da Cidade</label>
            <input asp-for="Nome" class="form-control" />
            <span asp-validation-for="Nome" class="text-danger"></span>
            <br />

            <input type="submit" value="Salvar dados" />
            <br />
            <span class="text-danger">@ViewBag.Erro</span>
        </form>
    </div>
</div>
```

Ajustando o menu principal e criando um link para nossos CRUDs

Como nosso projeto já possui 2 CRUDs, vamos ajustar o menu criando os links para ambos os cadastros. Também iremos criar uma página inicial e um controlador para ela.

Controller

```
public class HomeController : Controller
```

```
{
    public IActionResult Index()
    {
        return View();
    }
}
```

Crie a pasta Home para colocar o arquivo a seguir:

Home\Index.cshtml

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Bem vindo ao sistema de Gerenciamento de alunos e cidades!</h2>
<br>
<br>
<h3>Aqui há exemplos de como fazer um CRUD utilizando conceitos de herança</h3>
```

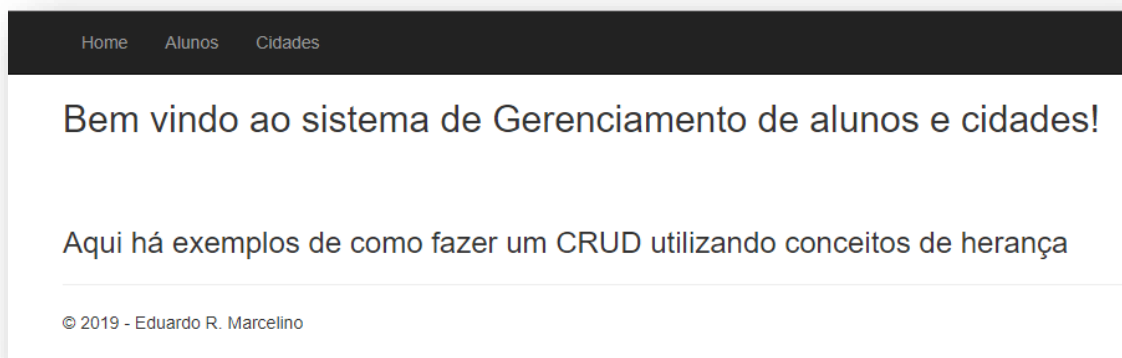
Ajuste o startup.cs:

```
template: "{controller=Home}/{action=Index}/{id?}");
```

Shared_Layout.cshtml

Ajuste os links a seguir:

```
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <ul class="nav navbar-nav">
                <li><a asp-area="" asp-controller="Home" asp-action="Index">Home</a></li>
                <li><a asp-area="" asp-controller="Aluno" asp-action="Index">Alunos</a></li>
                <li><a asp-area="" asp-controller="Cidade" asp-action="Index">Cidades</a></li>
            </ul>
        </div>
    </nav>
```



Exercícios

28.1 – Aplique os conceitos de herança no cadastro de Jogos.

29. Session

Fonte: <https://docs.microsoft.com/pt-br/aspnet/core/fundamentals/app-state?view=aspnetcore-3.0>
<https://www.devmedia.com.br/conceitos-e-dicas-usando-session/18104>

O estado de sessão do ASP.NET permite que você armazene e recupere valores para um usuário quando o usuário navega as diferentes páginas ASP.NET que compõem um aplicativo da Web. HTTP é um protocolo sem estado, o que significa que seu servidor Web trata cada solicitação HTTP para uma página como uma solicitação independente; por padrão, o servidor mantém nenhum conhecimento dos valores de variáveis usados durante solicitações anteriores. Como resultado, criar Aplicativos da Web que precisam manter algumas informações do estado cross-request (aplicativos que implementam carrinhos de compras, rolagem de dados e assim por diante) pode ser um desafio. O estado de sessão do ASP.NET identifica solicitações recebidas pelo mesmo navegador durante um intervalo de tempo limitado como uma sessão, e oferece uma maneira de persistir os valores de variáveis pela duração dessa sessão.

O ASP.NET Core mantém o estado de sessão fornecendo um cookie para o cliente que contém uma ID de sessão, que é enviada para o aplicativo com cada solicitação. O aplicativo usa a ID da sessão para buscar os dados da sessão.

Para utilizar o controle de sessão em uma aplicação ASP.NET CORE, edite o arquivo **startup.cs**, adicionando as linhas em destaque:

```
public void ConfigureServices(IServiceCollection services)
{
    ....

    services.AddSession(options =>
    {
        options.Cookie.IsEssential = true; // GDPR mais detalhes em https://andrewlock.net/session-state-gdpr-and-non-essential-cookies/
        options.IdleTimeout = TimeSpan.FromSeconds(1000000);
    });

    ....
}
```



```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    .....
    app.UseSession();
    .....
}
```

Nos controllers, adicione o namespace: `using Microsoft.AspNetCore.Http;`

Para armazenar um valor na sessão, utilize:

```
HttpContext.Session.SetString("Logado", "true");
```

Para carregar um valor da sessão, utilize:

```
string logado = HttpContext.Session.GetString("Logado");
```

30. Controle de acesso

A ideia do controle de acesso é basicamente não permitir que o usuário acesse áreas do site que ele só poderia se estivesse logado. Para tanto, iremos criar uma tela de login e, após validarmos se os dados de login conferem, iremos liberar o acesso às áreas restritas.

Como forma de centralizar esse controle e evitarmos que alguém tente “burlar” o controle de acesso, por exemplo digitando diretamente um link pelo navegador, iremos colocar o controle de acesso no controller base, e todos os controllers que herdarem dele ganharão de presente esta validação.

Vamos primeiro criar uma tela de login:

Crie uma pasta Login dentro de Views, e crie a view Index.cshtml:

`\Login\Index.cshtml`

```
@{
    ViewData["Title"] = "Login";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Login</h2>

<div class="row">
    <div class="col-lg-3">
        <form action="/login/Fazlogin" method="post">
            <label for="usuario" class="control-label">Usuário</label>
            <input type="text" name="usuario" class="form-control" required placeholder='Usuário...'/>
            <label for="senha" class="control-label">Senha</label>
```

```

        <input type="password" name="senha" class="form-control" required placeholder='Senha...'/>
        <br />
        <input type="submit" value="Login" class="btn btn-success"/>
        <br />
        <span class="text-danger"> @ViewBag.Erro</span>
    </form>
</div>
</div>

```

Crie dentro da pasta Controller a classe `HelperControllers`

HelperControllers

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace CadastroAlunoV1.Controllers
{
    public class HelperControllers
    {
        public static Boolean VerificaUserLogado(ISession session)
        {
            string logado = session.GetString("Logado");
            if (logado == null)
                return false;
            else
                return true;
        }
    }
}

```

Crie a classe para controlar o login:

LoginController

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace CadastroAlunoV1.Controllers
{
    public class LoginController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}

```

```

public IActionResult FazLogin(string usuario, string senha)
{
    //Este é apenas um exemplo, aqui você deve consultar na sua tabela de usuários
    //se existe esse usuário e senha
    if (usuario == "admin" && senha == "1234")
    {
        HttpContext.Session.SetString("Logado", "true");
        return RedirectToAction("index", "Home");
    }
    else
    {
        ViewBag.Erro = "Usuário ou senha inválidos!";
        return View("Index");
    }
}

public IActionResult LogOff()
{
    HttpContext.Session.Clear();
    return RedirectToAction("Index");
}
}
}

```

Modifique o PadraoController para que ele sempre verifique se o usuário está logado.

PadraoController

Adicione o atributo para informar se um determinado controller só deve ser acessado caso o usuário esteja autenticado. Caso algum controller possa ser acessado sem autenticação, basta no construtor colocar false nesta propriedade:

```

protected bool ExigeAutenticacao { get; set; } = true;

public override void OnActionExecuting(ActionExecutingContext context)
{
    if (ExigeAutenticacao && !HelperControllers.VerificaUserLogado(HttpContext.Session))
        context.Result = RedirectToAction("Index", "Login");
    else
    {
        ViewBag.Logado = true;
        base.OnActionExecuting(context);
    }
}

```

Bloqueie os links que não podem ser acessados sem autorização, e aproveite para incluir os links de **Login** e **Logoff**:

Shared_Layout.cshtml

```
<body>
```

```

<nav class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <ul class="nav navbar-nav">
      <li><a asp-area="" asp-controller="Home" asp-action="Index">Home</a></li>
      @if (ViewBag.Logado == true)
      {
        <li><a asp-area="" asp-controller="Aluno" asp-action="Index">Alunos</a></li>
        <li><a asp-area="" asp-controller="Cidade" asp-action="Index">Cidades</a></li>
        <li><a asp-area="" asp-controller="Login" asp-action="LogOff">Logoff</a></li>
      }
      else
      {
        <li><a asp-area="" asp-controller="Login" asp-action="Index">Login</a></li>
      }
    </ul>
  </div>
</nav>

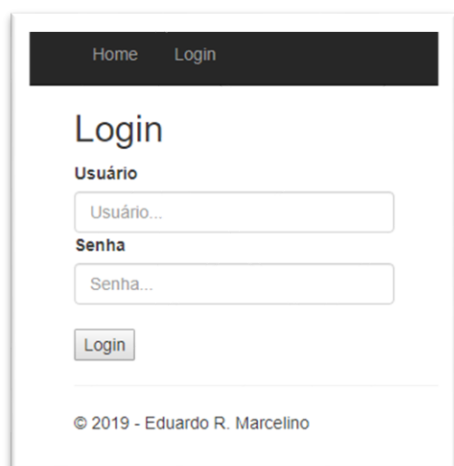
```

Altere a classe HomeController para que ela envie para o Html a informação de login do usuário. Observe que este controller não herda de PadraoController!

```

namespace CadastroAlunoV1.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            ViewBag.Logado = HelperControllers.VerificaUserLogado(HttpContext.Session);
            return View();
        }
    }
}

```



Home Alunos Cidades Logoff

Bem vindo ao sistema de Gerenciamento de alunos e cidades!

Aqui há exemplos de como fazer um CRUD utilizando conceitos de herança

© 2019 - Eduardo R. Marcelino

31. Salvando e Recuperando Imagens no Banco de Dados

Para exemplificar como salvar imagens, iremos utilizar o cadastro de imagens, incluindo um campo no banco de dados.

1. Alterações no banco de dados:

```
ALTER TABLE CIDADES ADD IMAGEM varbinary(max);
GO

ALTER procedure [dbo].[spUpdate_Cidades]
(
    @id int,
    @nome varchar(max) ,
    @imagem varbinary(max)
)
as
begin
    update cidades set nome = @nome , imagem=@imagem where id = @id
end
GO

ALTER procedure [dbo].[spInsert_Cidades]
(
    @id int,
    @nome varchar(max) ,
    @imagem varbinary(max)
)
as
begin
    insert into Cidades (id, nome, Imagem) values (@id, @nome , @imagem)
end
```

2. Alterações na ViewModel de Cidade:

```
using Microsoft.AspNetCore.Http;
using System;

namespace CadastroAlunoV1.Models
{
    public class CidadeViewModel : PadraoViewModel
```

```

{
    public string Nome { get; set; }

    /// <summary>
    /// Imagem recebida do form pelo controller
    /// </summary>
    public IFormFile Imagem { get; set; }

    /// <summary>
    /// Imagem em bytes pronta para ser salva
    /// </summary>
    public byte[] ImagemEmByte { get; set; }

    /// <summary>
    /// Imagem usada para ser enviada ao form no formato para ser exibida
    /// </summary>
    public string ImagemEmBase64
    {
        get
        {
            if (ImagemEmByte != null)
                return Convert.ToBase64String(ImagemEmByte);
            else
                return string.Empty;
        }
    }
}
}

```

3. Alterações no CidadeDAO

```

public class CidadeDAO : PadraoDAO<CidadeViewModel>
{
    protected override SqlParameter[] CriaParametros(CidadeViewModel model)
    {
        object imgByte = model.ImagemEmByte;
        if (imgByte == null)
            imgByte = DBNull.Value;

        SqlParameter[] parametros =
        {
            new SqlParameter("id", model.Id),
            new SqlParameter("nome", model.Nome),
            new SqlParameter("imagem", imgByte)
        };

        return parametros;
    }

    protected override CidadeViewModel MontaModel(DataRow registro)
    {
        CidadeViewModel c = new CidadeViewModel()
        {
            Id = Convert.ToInt32(registro["id"]),
            Nome = registro["nome"].ToString()
        };

        if (registro["imagem"] != DBNull.Value)
            c.ImagemEmByte = registro["imagem"] as byte[];
    }
}

```

```

        return c;
    }
    .....

```

4. Alterações no controller

```

public class CidadeController : PadraoController<CidadeViewModel>
{
    public CidadeController()
    {
        DAO = new CidadeDAO();
        GeraProximoId = true;
    }

    /// <summary>
    /// Converte a imagem recebida no form em um vetor de bytes
    /// </summary>
    /// <param name="file"></param>
    /// <returns></returns>
    public byte[] ConvertImageToByte(IFormFile file)
    {
        if (file != null)
            using (var ms = new MemoryStream())
            {
                file.CopyTo(ms);
                return ms.ToArray();
            }
        else
            return null;
    }

    protected override void ValidaDados(CidadeViewModel model, string operacao)
    {
        base.ValidaDados(model, operacao);
        if (string.IsNullOrEmpty(model.Nome))
            ModelState.AddModelError("Nome", "Preencha o nome.");

        //Imagem será obrigatio apenas na inclusão.
        //Na alteração iremos considerar a que já estava salva.
        if (model.Imagem == null && operacao == "I")
            ModelState.AddModelError("Imagem", "Escolha uma imagem.");

        if (model.Imagem != null && model.Imagem.Length / 1024 / 1024 >= 2)
            ModelState.AddModelError("Imagem", "Imagem limitada a 2 mb.");

        if (ModelState.IsValid)
        {
            //na alteração, se não foi informada a imagem, iremos manter a que já estava salva.
            if (operacao == "A" && model.Imagem == null)
            {
                CidadeViewModel cid = DAO.Consulta(model.Id);
                model.ImagemEmByte = cid.ImagemEmByte;
            }
            else
            {
                model.ImagemEmByte = ConvertImageToByte(model.Imagem);
            }
        }
    }
}

```

```
}
}
```

5. Alterações dentro do Form.cshtml da cidade, antes do botão salvar:

Altere a declaração do form, indicando que ele também manipula arquivos:

```
<form asp-action="Salvar" enctype="multipart/form-data">
```

Agora insira os elementos HTML para lidar com imagens, além do javascript que será usado para exibir uma prévia da figura:

```
<br />
<input type="file" name="Imagem" id="Imagem" onchange="exibirImagem()" class="btn btn-secondary">

<span asp-validation-for="Imagem" class="text-danger"></span>
```

No final do mesmo form, inclua o javascript abaixo. Ele tem por objetivo exibir uma pré-visualização da imagem na tela.

```
<script>
function exibirImagem() {
    var oFReader = new FileReader();
    oFReader.readAsDataURL(document.getElementById("Imagem").files[0]);
    oFReader.onload = function (oFREvent) {
        document.getElementById("imgPreview").src = oFREvent.target.result;
    };
}
</script>
```


[Home](#) [Alunos](#) [Cidades](#) [Logoff](#)

Cadastro de Cidade

RA

Nome do aluno

Visão_Panor...o_Campo.jpg



32. Formato JSON

<http://www.json.org/json-pt.html>
<https://pt.wikipedia.org/wiki/JSON>

Em computação, **JSON** (pronúncia ['dʒeɪzən], J-son em inglês), um acrônimo de JavaScript Object Notation, é um formato compacto, de padrão aberto independente, de troca de dados simples e rápida (parsing) entre sistemas, especificado por Douglas Crockford em 2000, que utiliza texto legível a humanos, no formato atributo-valor (natureza auto-descritiva). Isto é, um modelo de transmissão de informações no formato texto, muito usado em web services que usa transferência de estado representacional (REST) e aplicações AJAX, substituindo o uso do XML. O padrão foi especificado em 2000 e, definido em 2013 nos dois padrões concorrentes, [rfc:7159 RFC7159] e ECMA-404. Em 2017 a [rfc:8259 RFC8259] substituiu a 7159 e a ECMA-404 foi revisada.

O JSON é um formato de troca de dados entre sistemas independente de linguagem de programação derivado do JavaScript.^{[1][2]} Mas a partir de 2017 muitas linguagens de programação incluíram código para gerar, analisar sintaticamente dados em formato JSON e também converter para objetos da linguagem. O tipo de mídia da Internet oficial (MIME) para o JSON é `application/json` e nomes de arquivos JSON usam a extensão `.json`.

Exemplo de uma lista de alunos com suas notas

```
1 { "Alunos": [
2     { "nome": "Edson Sales Arantes", "notas": [ 8, 9, 5 ] },
3     { "nome": "Luiz Livelli ", "notas": [ 8, 10, 7 ] },
4     { "nome": "Caique Caicedo De Plata", "notas": [ 10, 10, 9 ] }
5 ] }
```

Em C#, para serializar um objeto, o comando é:

```
string carrinhoJson = JsonConvert.SerializeObject(carrinho);
```

Para deserializar:

```
List<CarrinhoViewModel> carrinho = JsonConvert.DeserializeObject<List<CarrinhoViewModel>>(carrinhoJson);
```

Obs: adicionar o namespace: `using Newtonsoft.Json;`

33. Carrinho de compras

Como exemplo de um carrinho de compras, faremos uma venda de terrenos por cidade. Segue abaixo as alterações necessárias.

6. Vamos criar uma model para armazenar os dados do carrinho na session.

```
public class CarrinhoViewModel : PadraoViewModel
{
    public int Quantidade { get; set; }
    public int CidadeId { get; set; }
    public string Nome { get; set; }
    public string ImagemEmBase64 { get; set; }
}
```

7. Crie um controller para o carrinho:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using CadastroAlunoV1.DA0;
using CadastroAlunoV1.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.AspNetCore.Mvc.Rendering;
using Newtonsoft.Json;

namespace CadastroAlunoV1.Controllers
{
    public class CarrinhoController : Controller
    {
        public IActionResult Index()
        {
            try
            {
                CidadeDAO dao = new CidadeDAO();
                var listaDeCidades = dao.Listagem();
                var carrinho = ObtemCarrinhoNaSession();
                // @ViewBag.TotalCarrinho = carrinho.Sum(c => c.Quantidade);
                @ViewBag.TotalCarrinho = 0;
                foreach (var c in carrinho)
                    @ViewBag.TotalCarrinho += c.Quantidade;

                return View(listaDeCidades);
            }
            catch (Exception erro)
            {
                return View("Error", new ErrorViewModel(erro.ToString()));
            }
        }
    }
}
```

```

public IActionResult Detalhes(int idCidade)
{
    try
    {
        List<CarrinhoViewModel> carrinho = ObtemCarrinhoNaSession();
        CidadeDAO cidDao = new CidadeDAO();
        var modelCidade = cidDao.Consulta(idCidade);
        CarrinhoViewModel carrinhoModel = carrinho.Find(c => c.CidadeId == idCidade);
        if (carrinhoModel == null)
        {
            carrinhoModel = new CarrinhoViewModel();
            carrinhoModel.CidadeId = idCidade;
            carrinhoModel.Nome = modelCidade.Nome;
            carrinhoModel.Quantidade = 0;
        }
        // preenche a imagem
        carrinhoModel.ImagemEmBase64 = modelCidade.ImagemEmBase64;
        return View(carrinhoModel);
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}

private List<CarrinhoViewModel> ObtemCarrinhoNaSession()
{
    List<CarrinhoViewModel> carrinho = new List<CarrinhoViewModel>();
    string carrinhoJson = HttpContext.Session.GetString("carrinho");
    if (carrinhoJson != null)
        carrinho = JsonConvert.DeserializeObject<List<CarrinhoViewModel>>(carrinhoJson);
    return carrinho;
}

public IActionResult AdicionarCarrinho(int CidadeId, int Quantidade)
{
    try
    {
        List<CarrinhoViewModel> carrinho = ObtemCarrinhoNaSession();
        CarrinhoViewModel carrinhoModel = carrinho.Find(c => c.CidadeId == CidadeId);
        if (carrinhoModel != null && Quantidade == 0)
        {
            //tira do carrinho
            carrinho.Remove(carrinhoModel);
        }
        else if (carrinhoModel == null && Quantidade > 0)
        {
            //não havia no carrinho, vamos adicionar
            CidadeDAO cidDao = new CidadeDAO();
            var modelCidade = cidDao.Consulta(CidadeId);
            carrinhoModel = new CarrinhoViewModel();
            carrinhoModel.CidadeId = CidadeId;
            carrinhoModel.Nome = modelCidade.Nome;
            carrinho.Add(carrinhoModel);
        }
        if (carrinhoModel != null)
            carrinhoModel.Quantidade = Quantidade;
        string carrinhoJson = JsonConvert.SerializeObject(carrinho);
        HttpContext.Session.SetString("carrinho", carrinhoJson);
        return RedirectToAction("Index");
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}

```

```

public IActionResult Visualizar()
{
    try
    {
        CidadeDAO dao = new CidadeDAO();
        var carrinho = ObtemCarrinhoNaSession();
        foreach (var item in carrinho)
        {
            var cid = dao.Consulta(item.CidadeId);
            item.ImagemEmBase64 = cid.ImagemEmBase64;
        }
        return View(carrinho);
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.ToString()));
    }
}

public override void OnActionExecuting(ActionExecutingContext context)
{
    if (!HelperControllers.VerificaUserLogado(HttpContext.Session))
        context.Result = RedirectToAction("Index", "Login");
    else
    {
        ViewBag.Logado = true;
        base.OnActionExecuting(context);
    }
}
}
}

```

8. Views. Crie a pasta \Views\Carrinho e adicione as views abaixo:

\Views\Carrinho\Index.cshtml

```

@model List<CidadeViewModel>
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<center><h2>Cidades com terrenos à venda</h2> </center>

<br />
<div class="fundo-vitrine">
    
    Há @ViewBag.TotalCarrinho terrenos no carrinho.
    <a href="/Carrinho/Visualizar">Visualizar</a>
</div>

<br />
<br />

<div class="fundo-vitrine">
    <ul class="ul-vitrine">
        @foreach (var cidade in Model)
        {
            <li class="li-vitrine">
                <a href="/Carrinho/Detalhes/?idCidade=@cidade.Id">
                    <figure class="figura-vitrine">

```

```

                
                <figcaption>
                    @cidade.Nome<br>
                </figcaption>
            </figure>
        </a>
    </li>
}
</ul>
</div>

@section Styles {
    <link href="@Url.Content("~/css/carrinho.css")" rel="stylesheet" type="text/css" />
}

```

\Views\Carrinho\Detalhes.cshtml

```

@model CarrinhoViewModel
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Escolha a quantidade de terrenos</h2>

<br />
<br />
<div class="row">
    <div class="col-md-4">
        <form asp-action="AdicionarCarrinho">
            <input asp-for="CidadeId" hidden/>

            
            <br />
            <strong>Cidade : @Model.CidadeId - @Model.Nome</strong>
            <br />
            <br />
            Quantidade:
            <input asp-for="Quantidade"/>

            <br />
            <br />
            <input type="submit" value="Adicionar ao carrinho" class="btn btn-primary" />

            <br />
            <span class="text-danger">@ViewBag.Erro</span>
        </form>
    </div>
</div>

```

Views\Carrinho\Visualizar.cshtml

```

@model List<CarrinhoViewModel>
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Terrenos no carrinho</h2>

<br />

<table class="table table-responsive">
    <tr>
        <th>Cidade</th>
        <th>Imagem</th>
        <th>Quantidade</th>
    </tr>

    @foreach (var item in Model)
    {
        <tr>
            <td>@item.CidadeId - @item.Nome</td>
            <td></td>
            <td>@item.Quantidade</td>
        </tr>
    }
</table>

<br />
<a href="/Carrinho/Index">Retornar para as compras</a>
<br />
<br />
<a href="/Carrinho/EfetuarPedido">Fechar pedido!!!</a>

```

9. Vamos criar um arquivo de CSS em:

wwwroot\css\carrinho.css

```

.linkAcao:hover {
    background-color: #3333ff;
}

.linkAcao:visited {
    text-decoration: none;
}

.linkAcao:active {
    text-decoration: none;
}

.linkAcao:link {
    text-decoration: none;
}

.figura-vitrine {
    width: 190px;
    height: 332px;
    margin: 0;
    padding: 0;
    box-shadow: 3px 3px 3px black;
}

.figura-vitrine:hover {
    box-shadow: 7px 7px 7px red;
}

.imagem-vitrine {
    height: inherit;
    width: inherit;
}

```

```

.figura-vitrine figcaption {
    margin-top: 5px;
}

.ul-vitrine {
    list-style: none;
}

.li-vitrine {
    display: inline-block;
    width: 200px;
    height: 410px;
    text-align: center;
    font-size: small;
    vertical-align: top;
}

.fundo-vitrine {
    background-color: #cccccc;
    width: 900px;
    padding: 5px;
    margin: 0 auto;
    border-radius: 10px;
}

margin-bottom: 10px;
}

```

10. Alterações no menu (Layout.cshtml)

```

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Sistema de gerenciamento de alunos e cidades</title>

    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
    <link rel="stylesheet" href="~/css/site.css" />

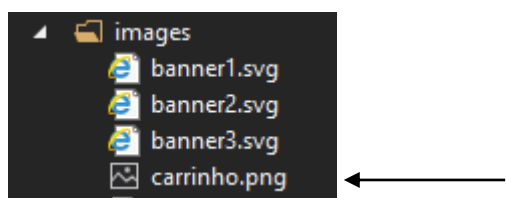
    @*essa parte abaixo é para permitir adicionar um css no head a partir de qualquer página*@
    @RenderSection("styles", false)
</head>

.....

@if (ViewBag.Logado == true)
{
    <li><a asp-area="" asp-controller="Aluno" asp-action="Index">Alunos</a></li>
    <li><a asp-area="" asp-controller="Cidade" asp-action="Index">Cidades</a></li>
    <li><a asp-area="" asp-controller="Carrinho" asp-action="Index">Compra de cidades</a></li>
    <li><a asp-area="" asp-controller="Login" asp-action="LogOff">Logoff</a></li>
}
else
{
    <li><a asp-area="" asp-controller="Login" asp-action="Index">Login</a></li>
}

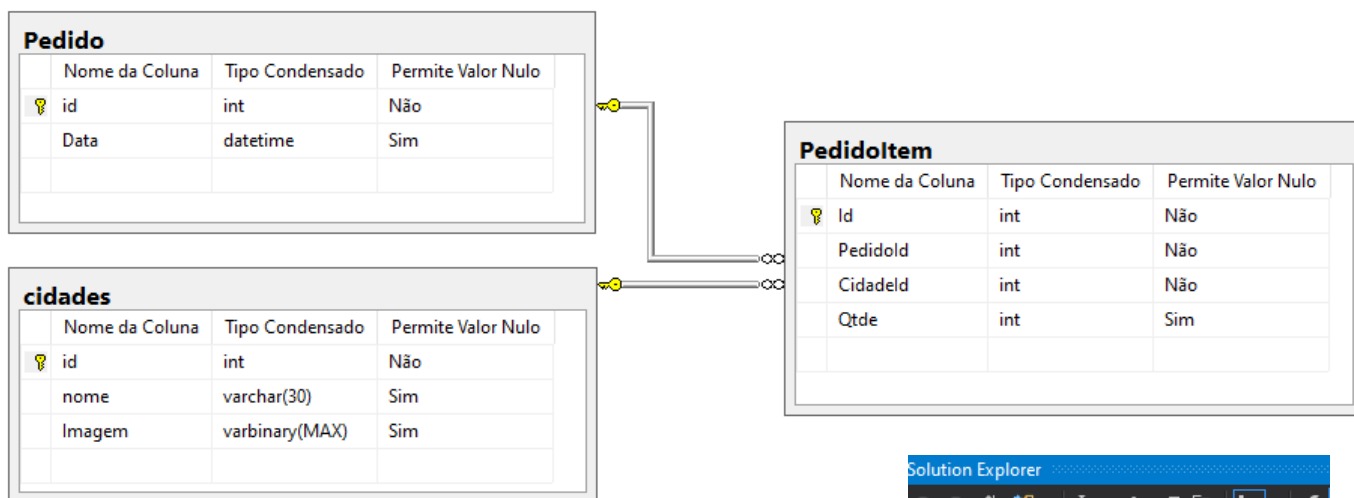
```

Uma figura representando um carrinho  foi adicionada ao projeto, na pasta abaixo:



34. Desenvolvimento de um cadastro Mestre Detalhe

1. Alterações no banco de dados:

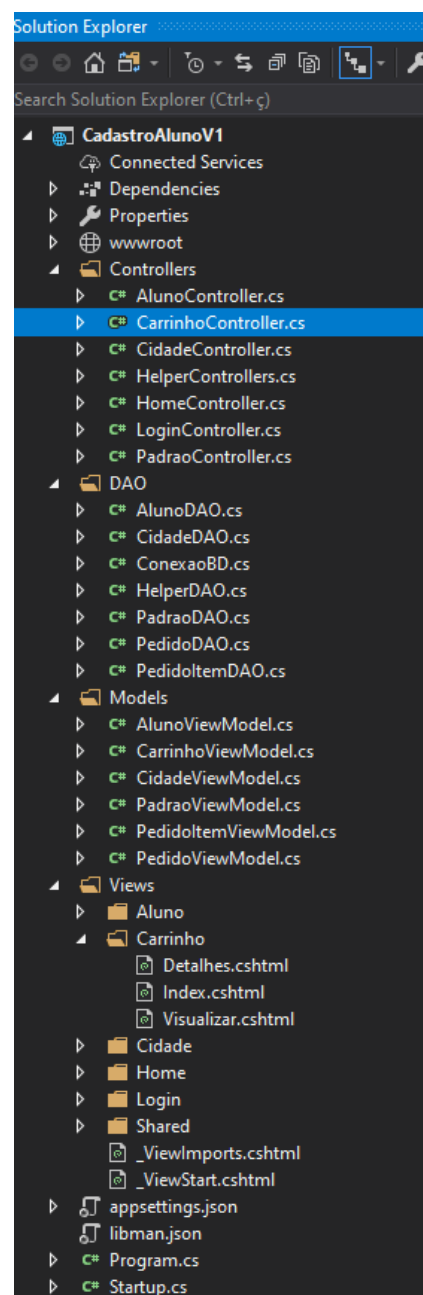


```
CREATE TABLE[dbo].[Pedido]
(
    [id] [int] IDENTITY(1,1) NOT NULL primary key,
    [Data] [datetime] NULL
)
GO
```

```
CREATE TABLE[dbo].[PedidoItem]
(
    [Id][int] IDENTITY(1,1) NOT NULL primary key,
    [PedidoId] [int] NOT NULL,
    [CidadeId] [int] NOT NULL,
    [Qtde] [int] NULL,
)
GO
```

```
create procedure spInsert_Pedido
    @id int,
    @data datetime
as
begin
    insert into pedido(data) values (@data)
end
GO
```

```
create procedure spInsert_PedidoItem
    @id int,
    @PedidoId int,
    @CidadeId int,
    @Qtde int
as
begin
    insert into pedidoItem(pedidoId, cidadeId, Qtde)
    values (@PedidoId, @CidadeId, @Qtde)
end
GO
```



2. Modificações na classe **HelperDAO**

O objetivo da modificação a seguir é retornar o último id inserido em campos identity.

```
public static int ExecutaProc(string nomeProc,
                             SqlParameter[] parametros,
                             bool consultaUltimoIdentity = false)
{
    using (SqlConnection conexao = ConexaoBD.GetConexao())
    {
        using (SqlCommand comando = new SqlCommand(nomeProc, conexao))
        {
            comando.CommandType = CommandType.StoredProcedure;
            if (parametros != null)
                comando.Parameters.AddRange(parametros);
            comando.ExecuteNonQuery();

            if (consultaUltimoIdentity)
            {
                string sql = "select isnull(@@IDENTITY,0)";
                comando.CommandType = CommandType.Text;
                comando.CommandText = sql;
                int pedidoId = Convert.ToInt32(comando.ExecuteScalar());
                conexao.Close();
                return pedidoId;
            }
            else
                return 0;
        }
    }
}
```

3. Modificações no **PadraoDAO**

O objetivo desta modificação é retornar o último id inserindo.

```
protected bool ChaveIdentity { get; set; } = false;

public virtual int Insert(T model)
{
    return HelperDAO.ExecutaProc("spInsert_" + Tabela, CriaParametros(model), ChaveIdentity);
}
```

4. Novas ViewModels para Pedido e PedidoItem

```
public class PedidoViewModel : PadraoViewModel
{
    public DateTime Data { get; set; }
}

public class PedidoItemViewModel : PadraoViewModel
{
    public int PedidoId { get; set; }
    public int CidadeId { get; set; }
    public int Qtde { get; set; }
}
```

5. Classe **PedidoDAO**

```
using CadastroAlunoV1.Models;
using System;
using System.Data;
using System.Data.SqlClient;

namespace CadastroAlunoV1.DAO
{
    public class PedidoDAO : PadraoDAO<PedidoViewModel>
    {
        protected override SqlParameter[] CriaParametros(PedidoViewModel model)
        {
            SqlParameter[] parametros =
            {
                new SqlParameter("id", model.Id),
                new SqlParameter("data", model.Data)
            };
            return parametros;
        }

        protected override PedidoViewModel MontaModel(DataRow registro)
        {
            PedidoViewModel c = new PedidoViewModel()
            {
                Id = Convert.ToInt32(registro["id"]),
                Data = Convert.ToDateTime(registro["data"])
            };

            return c;
        }

        protected override void SetTabela()
        {
            Tabela = "Pedido";
            ChaveIdentity = true;
        }
    }
}
```

6. Classe **PedidoItemDAO**

```
public class PedidoItemDAO : PadraoDAO<PedidoItemViewModel>
{
    protected override SqlParameter[] CriaParametros(PedidoItemViewModel model)
    {
        SqlParameter[] parametros =
        {
            new SqlParameter("id", model.Id),
            new SqlParameter("PedidoId", model.PedidoId),
            new SqlParameter("CidadeId", model.CidadeId),
            new SqlParameter("Qtde", model.Qtde)
        };

        return parametros;
    }
}
```

```

protected override PedidoItemViewModel MontaModel(DataRow registro)
{
    PedidoItemViewModel c = new PedidoItemViewModel()
    {
        Id = Convert.ToInt32(registro["id"]),
        CidadeId = Convert.ToInt32(registro["Cidadeid"]),
        PedidoId = Convert.ToInt32(registro["PedidoId"]),
        Qtde = Convert.ToInt32(registro["id"]),
    };

    return c;
}

protected override void SetTabela()
{
    Tabela = "PedidoItem";
    ChaveIdentity = true;
}
}

```

7. Adicione o seguinte método na classe CarrinhoController

```

public class CarrinhoController : Controller
{
    public IActionResult EfetuarPedido()
    {
        try
        {
            using (var transacao = new System.Transactions.TransactionScope())
            {
                PedidoViewModel pedido = new PedidoViewModel();
                pedido.Data = DateTime.Now;
                PedidoDAO pedidoDAO = new PedidoDAO();
                int idPedido = pedidoDAO.Insert(pedido);
                PedidoItemDAO itemDAO = new PedidoItemDAO();
                var carrinho = ObtemCarrinhoNaSession();
                foreach (var elemento in carrinho)
                {
                    PedidoItemViewModel item = new PedidoItemViewModel();
                    item.PedidoId = idPedido;
                    item.CidadeId = elemento.CidadeId;
                    item.Qtde = elemento.Quantidade;
                    itemDAO.Insert(item);
                }
                transacao.Complete();
            }

            HelperControllers.LimparCarrinho(HttpContext.Session);

            return RedirectToAction("Index", "Home");
        }
        catch (Exception erro)
        {
            return View("Error", new ErrorViewModel(erro.ToString()));
        }
    }
}

```

35. AJAX

AJAX é o acrônimo para **JavaScript assíncrono + XML**. Não é exatamente uma tecnologia nova, mas um termo empregado em 2005 por Jesse James Garrett para descrever uma nova forma de utilizar em conjunto algumas tecnologias, incluindo HTML ou XHTML, CSS, JavaScript, DOM, XML, XSLT, e o mais importante: objeto XMLHttpRequest.

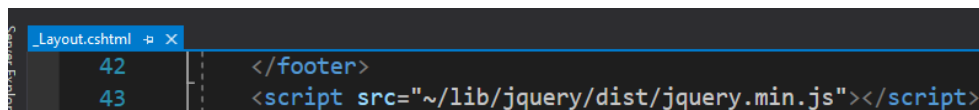
Quando essas tecnologias são combinadas no modelo AJAX, as aplicações web que a utilizam são capazes de fazer rapidamente atualizações incrementais para a interface do usuário sem recarregar a página inteira do navegador. Isso torna a aplicação mais rápida e sensível às ações do usuário.

Embora a letra X em AJAX corresponda ao XML, atualmente o JSON é mais utilizado que o XML devido às suas vantagens, como ser mais leve e ser parte do JavaScript. Ambos (JSON e XML) são utilizados para obter informações do pacote no modelo AJAX. Fonte: <https://developer.mozilla.org/pt-BR/docs/Web/Guide/AJAX>

Uma forma simples de utilizar o AJAX é utilizando a biblioteca JQuery (<https://jquery.com/>). Esta biblioteca permite a execução de chamadas AJAX de forma simples e rápida, além de ser o padrão adotado pelo mercado para trabalhar com esta tecnologia. Os projetos feitos em Asp.NET Core MVC já trazem consigo a biblioteca do JQuery configurada.

Para utilizar o JQuery você precisará adicionar uma instrução para o carregamento da biblioteca Javascript. A linha abaixo, como dito anteriormente, já está configurada nos projetos padrão do visual studio, no arquivo Layout.cshtml.

```
<script src="~/lib/jquery/dist/jquery.min.js"></script>
```



Aplicando AJAX ao projeto de soma de valores

No capítulo 5 fizemos um projeto para efetuar a soma de dois valores. Vamos fazer uma alteração no projeto para que a soma seja efetuada através de uma chamada AJAX.

No projeto original, quando o usuário pressionava o botão [Somar] o formulário era submetido ao servidor, que efetuava os cálculos e desenhava novamente a página toda preenchendo a resposta. Observe que os campos com os valores preenchidos pelo usuário foram apagados. Isso ocorre porque nós não programamos para que eles fossem preenchidos novamente ao desenhar a tela novamente. Vamos mudar as coisas com AJAX!

1 – Na HomeController, vamos criar um novo método para efetuar o soma através de uma chamada AJAX:

```
public IActionResult EfetuaSomaAjax(int valor1, int valor2)
{
    int resultado = valor1 + valor2;
    return Json(new { soma = resultado });
}
```

Observe que a resposta é um Json, que está devolvendo um objeto **anônimo**¹. O objeto em questão possui apenas um atributo (soma) cujo valor será o resultado da soma dos valores. Este objeto será enviado para a página e será manipulado por código javascript.

2 – No arquivo site.js vamos criar uma function para a chamada:

```
function efetuaSoma() {

    var valor1 = document.getElementById("valor1").value;
    var valor2 = document.getElementById("valor2").value;

    var linkAPI = '/home/EfetuaSomaAjax?valor1=' + valor1 + '&valor2=' + valor2;

    $.ajax({
        url: linkAPI,
        success: function (dados) {
            document.getElementById("resultado").value = dados.soma;
        }
    });
}
```

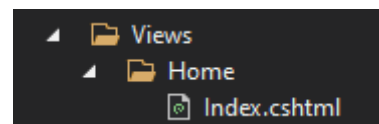


No código acima, \$ é a forma de acessar os métodos JQuery (dentre os vários, temos o método ajax). Na chamada do método ajax temos como parâmetro a **url** que é o endereço ao qual estamos submetendo nossa chamada e também o call-back (evento) **success**, que será disparado em caso de sucesso da nossa chamada. O parâmetro **dados** recebe o nosso objeto enviado em Json pelo C#.

3- Agora vamos alterar o form principal para que a função Js acima seja chamada ao se pressionar o botão.

Ajuste o Index.cshtml colocando a identificação nos objetos e a chamada ao método efetuaSoma().

```
<form asp-action="EfetuaSoma">
    Valor 1:
    <input type="number" id="valor1" />    <br />
    Valor 2:
    <input type="number" id="valor2" />    <br><br>
    <input type="button" value="Somar" onclick="efetuaSoma()" />    <br /> <br />
    Resultado:
    <input type="number" id="resultado" disabled />
</form>
```



Observe que agora os campos preenchidos pelo usuário não serão apagados ao se efetuar a soma pois com AJAX apenas os valores são submetidos ao servidor e apenas o conteúdo do campo Resultado é alterado no retorno.

Pressione CTRL + F5 no seu navegador web caso as mudanças efetuadas no seu código javascript não pareçam surtir efeito. Isso força o carregamento do Js pelo navegador web. Para ver se há algum erro ocorrendo com o JS, pressione F12.

¹ Os tipos anônimos fornecem um modo conveniente de encapsular um conjunto de propriedades somente leitura em um único objeto sem a necessidade de primeiro definir explicitamente um tipo. Veja mais em: [Tipos anônimos | Microsoft Docs](#)

Consumindo uma API via AJAX (API de consulta de CEP)

A ViaCEP fornece de forma gratuita a consulta a sua base de CEP. Iremos a seguir desenvolver um projeto onde iremos permitir que o usuário digite um CEP e iremos então submeter este CEP à API disponibilizada por este site, que irá retornar os dados do CEP informado.

A imagem abaixo representa uma consulta à esta API feita diretamente no navegador. Observe no endereço abaixo que a parte **azul** refere-se ao CEP consultado. No final da URL especificamos que queremos o retorno no formato JSON. O endereço consultado foi o seguinte: <https://viacep.com.br/ws/09030390/json/>



Crie um novo projeto do tipo Asp.net MVC igual aos demais já criados anteriormente. Lembre-se que estes projetos criados no Visual Studio já são acompanhados do JQuery e precisaremos dele!

Substitua o conteúdo do **index.html** pelo código a seguir:

```
<form>
  CEP: <br>
  <input type="text" name="cep" id="cep" />
  <input type="button" name="btnBuscaCEP" value="Buscar" onclick="buscaCEP()" />

  <br>
  Logradouro: <br>
  <input type="text" name="logradouro" id="logradouro" /> <br>

  Número: <br>
  <input type="text" name="numero" id="numero" />    <br>

  Complemento: <br>
  <input type="text" name="complemento" id="complemento" />    <br>

  Bairro: <br>
  <input type="text" name="bairro" id="bairro" />    <br>
  Cidade: <br>

  <input type="text" name="localidade" id="localidade" />    <br>
  Estado: <br>

  <input type="text" name="uf" id="uf" />    <br>
</form>
```



Quando o botão de busca de CEP for pressionado ele irá executar a function javascript buscaCEP(). Para tanto, iremos colocar esta function no site.js:

```
function buscaCEP() {

    var cep = document.getElementById("cep").value;
    cep = cep.replace('-', ''); // removemos o traço do CEP

    if (cep.length > 0) {
        var linkAPI = 'https://viacep.com.br/ws/' + cep + '/json/';

        $.ajax({
            url: linkAPI,

            beforeSend: function () {
                document.getElementById("logradouro").value = '...';
                document.getElementById("bairro").value = '...';
                document.getElementById("localidade").value = '...';
                document.getElementById("uf").value = '...';
            },

            success: function (dados) {
                if (dados.erro != undefined) // quando o CEP não existe...
                {
                    alert('CEP não localizado...');
                    document.getElementById("logradouro").value = '';
                    document.getElementById("bairro").value = '';
                    document.getElementById("localidade").value = '';
                    document.getElementById("uf").value = '';
                }
                else // quando o CEP existe
                {
                    document.getElementById("logradouro").value = dados.logradouro;
                    document.getElementById("bairro").value = dados.bairro;
                    document.getElementById("localidade").value = dados.localidade;
                    document.getElementById("uf").value = dados.uf;
                }
            }
        });
    }
}
```



Observe que a chamada AJAX acima possui um novo evento: **beforeSend**. Este evento é disparado antes que a chamada aconteça. Quando o CEP não é localizado, o atributo **erro** é retornado e ele é o indicativo que precisamos para saber que o CEP não existe. A documentação foi fornecida pelo próprio ViaCep.

CEP:

Logradouro:

Número:

Complemento:

Bairro:

Cidade:

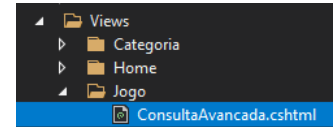
Estado:

Construindo uma consulta avançada de dados com AJAX

Avançando no uso do AJAX, iremos construir uma tela para efetuar consulta de registros no banco de dados. Para isso, iremos utilizar o nosso projeto do cadastro de jogos na sua última versão (que implementa os conceitos de herança).

1 - Crie uma nova página na pasta Jogo : **ConsultaAvancada.cshtml**.

2 - O código da página será o seguinte:



```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<fieldset id="areaFiltro" class="form-group">
    <legend>Consulta avançada de Jogos</legend>
    <div class="row">
        <div class="col-lg-4">
            Descrição<br />
            <input type="text" id="descricao" class="form-control" />
        </div>
        <div class="col-lg-3">
            Categoria <br />
            <select id="categoria" class="form-control" asp-items="@ViewBag.Categorias"></select>
        </div>
        <div class="col-lg-2">
            Período <br />
            <input type="date" id="dataInicial" class="form-control" />
        </div>
        <div class="col-lg-2">
            <br />
            <input type="date" id="dataFinal" class="form-control" />
        </div>
        <div class="col-lg-1">
            <br />
            <input type="button" id="btnFiltro" class="btn btn-success" value="Aplicar"
onclick="aplicaFiltroConsultaAvancada()" />
        </div>
    </div>
</fieldset>

<div id="resultadoConsulta" class="table-responsive">
</div>
```

- Observações sobre o Código acima:
 - Veja que a caixa de seleção de categorias é preenchida da mesma forma que fizemos no cadastro de jogos;
 - Observe o evento onclick no botão de aplicar filtro: ele está executando uma função Javascript que, via AJAX, irá atualizar a tela exibindo o grid com os resultados da consulta;
 - Utilizamos o Sistema de grid do Bootstrap para organizar as informações;
 - A div **resultadoConsulta** está propositamente vazia pois será ali que os dados da consulta serão “injetados”.

3 - Para um toque de frescura, coloque estas classes de css no **site.css**:

```
fieldset {
    margin: 20px;
    padding: 0 10px 10px;
    border: 1px solid #666;
    border-radius: 8px;
    padding-top: 10px;
}

legend {
    padding: 2px 4px;
}
```

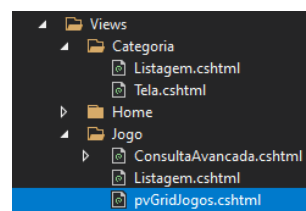


4 - Crie outra página em branco na pasta Jogo chamada: **pvGridJogos.cshtml**

Esta página será processada como uma partial view, que é uma página que é processada pelo controller e seu conteúdo será devolvido no formato HTML para posteriormente ser “injetado” na página via javascript.

@model List<JogoViewModel>

```
<table class="table table-striped">
    <tr>
        <th>Código</th>
        <th>Nome</th>
        <th>Data de lançamento</th>
        <th>Categoria</th>
        <th>Valor</th>
    </tr>
    @foreach (var jogo in Model)
    {
        <tr>
            <td>@jogo.Id</td>
            <td>@jogo.Descricao</td>
            <td>@jogo.DataAquisicao.ToShortDateString()</td>
            <td>@jogo.CategoriaID - @jogo.DescricaoCategoria</td>
            <td>@jogo.Valor.ToString("0.00")</td>
        </tr>
    }
</table>
```



5 – Para que a magia aconteça, é necessário um pouco de javascript. Adicione no site.js a function abaixo para fazer a chamada AJAX:

```
function aplicaFiltroConsultaAvancada() {

    var vDescricao = document.getElementById('descricao').value;
    var vCategoria = document.getElementById('categoria').value;
    var vDataInicial = document.getElementById('dataInicial').value;
    var vDataFinal = document.getElementById('dataFinal').value;

    $.ajax({
        url: "/jogo/ObtemDadosConsultaAvancada",
        data: { descricao: vDescricao, categoria: vCategoria, dataInicial: vDataInicial, dataFinal: vDataFinal },
        success: function (dados) {
            if (dados.erro != undefined) {
                alert(dados.msg);
            }
            else {
                document.getElementById('resultadoConsulta').innerHTML = dados;
            }
        },
    });
}
```

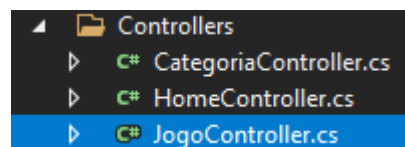
Observe que na chamada acima os parâmetros são enviados através do parâmetro **data:** e no formato **JSON**. Este é um jeito diferente de passar os parâmetros e poderíamos ter feito assim nos exemplos anteriores. **Veja que os nomes dos campos do JSON devem coincidir com os nomes dos parâmetros do método no controller!!!!**

6 – Adicione no _Layout.cshtml uma chamada para esta consulta:



```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Jogo" asp-action="ExibeConsultaAvancada">Consulta
    Avançada de Jogos</a>
</li>
```

7 – Agora, precisamos criar os métodos no controller Jogo para exibir a tela de consulta avançada e também para aplicar o filtro:



```
public IActionResult ExibeConsultaAvancada()
{
    try
    {
        PreparaComboCategorias();
        ViewBag.Categorias.Insert(0, new SelectListItem("TODAS", "0"));
        return View("ConsultaAvancada");
    }
    catch (Exception erro)
    {
        return View("Error", new ErrorViewModel(erro.Message));
    }
}
```

```

public IActionResult ObtemDadosConsultaAvancada(string descricao,
                                                int categoria,
                                                DateTime dataInicial,
                                                DateTime dataFinal)
{
    try
    {
        JogoDAO dao = new JogoDAO();
        if (string.IsNullOrEmpty(descricao))
            descricao = "";

        if (dataInicial.Date == Convert.ToDateTime("01/01/0001"))
            dataInicial = SqlDateTime.MinValue.Value;

        if (dataFinal.Date == Convert.ToDateTime("01/01/0001"))
            dataFinal = SqlDateTime.MaxValue.Value;

        var lista = dao.ConsultaAvancadaJogos(descricao, categoria, dataInicial, dataFinal);
        return PartialView("pvGridJogos", lista);
    }
    catch (Exception erro)
    {
        return Json(new { erro = true, msg = erro.Message });
    }
}

private void PreparaComboCategorias()
{
    CategoriaDAO dao = new CategoriaDAO();
    var lista = dao.Listagem();

    List<SelectListItem> listaRetorno = new List<SelectListItem>();
    foreach (var categ in lista)
        listaRetorno.Add(new SelectListItem(categ.Descricao, categ.Id.ToString()));

    ViewBag.Categorias = listaRetorno;
}

```

8- Crie a stored Procedure que será a responsável por selecionar os dados com base no filtros de tela.

```

create procedure [dbo].[spConsultaAvancadaJogos]
(
    @descricao varchar(max),
    @categoria int,
    @dataInicial datetime,
    @dataFinal datetime)
as
begin
    declare @categIni int
    declare @categFim int

    set @categIni = case @categoria when 0 then 0           else @categoria end
    set @categFim = case @categoria when 0 then 999999      else @categoria end

    select jogos.*, Categorias.descricao as 'DescricaoCategoria'
    from Jogos
    inner join Categorias on jogos.categoriaId = categorias.id
    where jogos.descricao like '%' + @descricao + '%' and
           jogos.data_aquisicao between @dataInicial and @dataFinal and
           jogos.categoriaID between @categIni and @categFim;
end

```

9 – Adicione no Jogo.DAO:

```
public List<JogoViewModel> ConsultaAvancadaJogos(string descricao,
                                                int categoria,
                                                DateTime dataInicial,
                                                DateTime dataFinal)
{
    SqlParameter[] p = {
        new SqlParameter("descricao", descricao),
        new SqlParameter("categoria", categoria),
        new SqlParameter("dataInicial", dataInicial),
        new SqlParameter("dataFinal", dataFinal),
    };

    var tabela = HelperDAO.ExecutaProcSelect("spConsultaAvancadaJogos", p);
    var lista = new List<JogoViewModel>();
    foreach (DataRow dr in tabela.Rows)
        lista.Add(MontaModel(dr));

    return lista;
}
```

Ajuste o método abaixo:

```
public static JogoViewModel MontaModel(DataRow registro)
{
    JogoViewModel Jogo = new JogoViewModel();
    Jogo.Id = Convert.ToInt32(registro["id"]);
    Jogo.Descricao = registro["descricao"].ToString();
    Jogo.CategoriaID = Convert.ToInt32(registro["categoriaID"]);
    Jogo.Valor = Convert.ToDouble(registro["valor_locacao"]);
    Jogo.DataAquisicao = Convert.ToDateTime(registro["data_aquisicao"]);

    if (registro.Table.Columns.Contains("DescricaoCategoria"))
        Jogo.DescricaoCategoria = registro["DescricaoCategoria"].ToString();

    return Jogo;
}
```

10 - Ufa! Agora é hora de executar e ver o resultado final:

Consulta avançada de Jogos

Descrição

Categoria

TODAS

Período

dd/mm/aaaa



dd/mm/aaaa



Aplicar

Código	Nome	Data de lançamento	Categoria	Valor
1	mario kart double dash	01/01/1991	3 - Corrida	61.00
2	Naruto o ninja pokemon	01/01/1996	1 - Ação	62.00
3	Pokemon	01/01/1999	2 - RPG	33.00
4	street fighter II	01/01/1989	1 - Ação	55.00
5	FIFA	23/03/2021	1 - Ação	6.00
6	PES	20/04/2021	2 - RPG	7.98
7	Counter Strike	01/01/2017	5 - Tiro	9.63
8	Far cry	01/01/2017	5 - Tiro	7.99
9	Call of duty	01/01/2017	5 - Tiro	2.55
10	Doom 2	03/06/2021	5 - Tiro	1.30

36. Consumindo APIs

Nós fizemos no capítulo anterior o consumo de APIs utilizando AJAX, agora o faremos via controller. O exemplo a seguir poderia ter sido feito totalmente apenas com AJAX, porém fizemos o consumo da API de dentro da controller apenas para exemplificar.

1 – Crie um novo projeto

2 – Apague totalmente o conteúdo do **index.cshtml** da Home e substitua-a por:

```
@{
    ViewData["Title"] = "Home Page";
}

<h2 class="text-center">Consulta de endereço utilizando Chamada de API</h2>

<table class="table table-striped">
    <caption></caption>
    <tr>
        <td>Estado</td>
        <td><input type="text" id="estado" maxlength="2" value="SP" class="form-control" /></td>
    </tr>
    <tr>
        <td>Cidade</td>
        <td><input type="text" id="cidade" value="São Bernardo do Campo" class="form-control" /></td>
    </tr>
    <tr>
        <td>Rua/Avenida</td>
        <td><input type="text" id="rua" class="form-control" /> </td>
    </tr>
    <tr>
        <td colspan="2" align="center">
            <input type="button" value="Pesquisar" onclick="consultar()" class="btn btn-success" />
        </td>
    </tr>
</table>

<br />

<div id="resultado">
</div>
```

3 – Crie o seguinte function no **site.js**:

```
function consultar() {
    var estado = $("#estado").val();
    var cidade = $("#cidade").val();
    var rua = $("#rua").val();

    $.ajax({
        url: "/home/ConsultaEndereco",
        data: { estado: estado, cidade: cidade, rua: rua },
        dataType: "json",
        success: function (resultados) {
            if (resultados.erro != undefined) {
                alert(resultados.msg);
            }
            else {
                var html = "<table class='table table-striped'>";
                html += "<tr>";
                html += "<th>CEP</th> <th>Logradouro</th> <th>Bairro</th>";
                html += "</tr>";

                for (var n = 0; n < resultados.length; n++) {
                    html += "<tr>";
```




```

        html += "<td>" + resultados[n].cep + "</td>"
        html += "<td>" + resultados[n].logradouro + "</td>"
        html += "<td>" + resultados[n].bairro + "</td>"
        html += "</tr>";
    }

    html += "</table>";

    $("#resultado").html(html);
}
});
}
}

```

4 – Adicione o seguinte método ao **HomeController**:

```

public IActionResult ConsultaEndereco(string estado, string cidade, string rua)
{
    try
    {
        var proxy = new WebProxy
        {
            Address = new Uri("http://proxycefsa.cefsa.corp.local:8080"),
            BypassProxyOnLocal = true,
            UseDefaultCredentials = true, //não informaremos usuário e senha
        };

        var handler = new HttpClientHandler();
        handler.Proxy = proxy;

        using (var httpClient = new HttpClient(handler))
        {
            string url = $"http://viacep.com.br/ws/{estado}/{cidade}/{rua}/json/";
            using (var response = httpClient.GetAsync(url).Result)
            {
                if (response.StatusCode == System.Net.HttpStatusCode.OK)
                {
                    string resposta = response.Content.ReadAsStringAsync().Result;
                    return Content(resposta);
                }
                else
                {
                    throw new Exception("Erro ao consultar. Code: " + response.StatusCode);
                }
            }
        }
    }
    catch (Exception erro)
    {
        return Json(new { erro = true, msg = erro.Message });
    }
}

```

Consulta de endereço utilizando Chamada de API

Estado	<input type="text" value="SP"/>	
Cidade	<input type="text" value="São Bernardo do Campo"/>	
Rua/Avenida	<input type="text" value="Alvarenga"/>	

CEP	Logradouro	Bairro
09840-650	Rua Alvarenga Peixoto	Dos Casa
09850-550	Estrada dos Alvarengas	Assunção