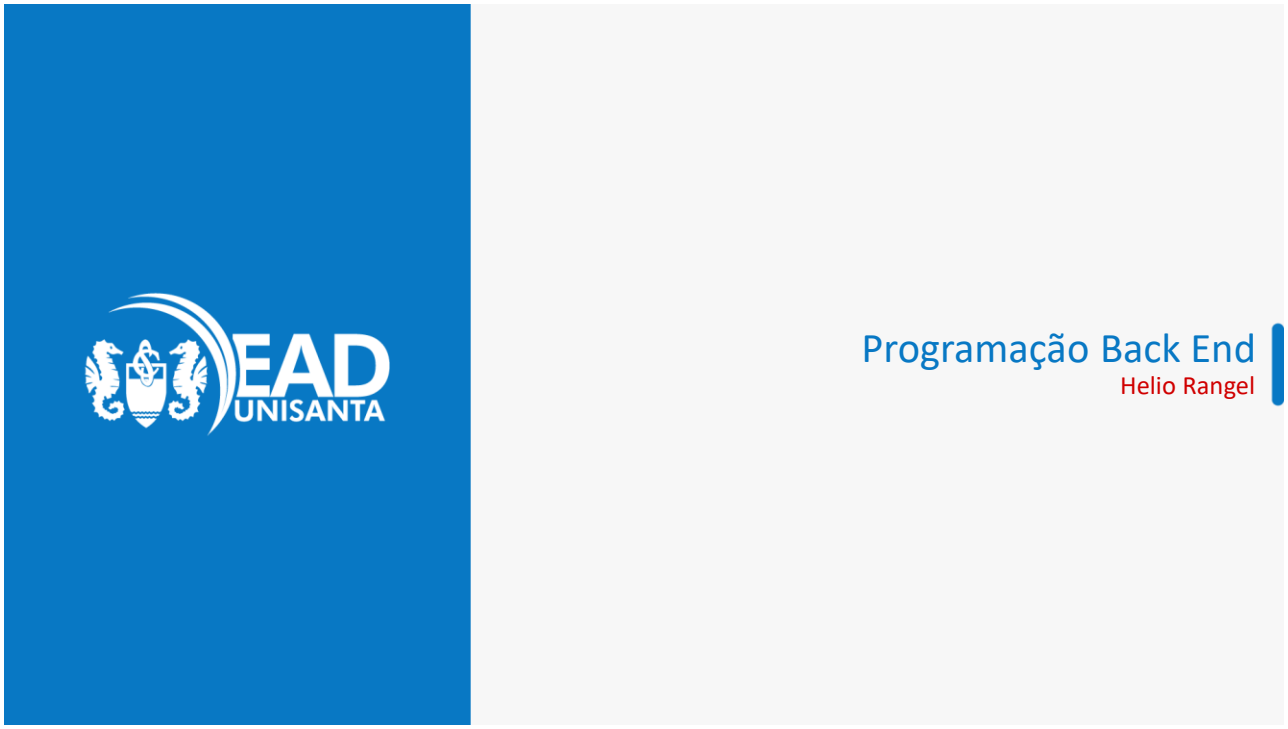
1



Programação Back End

Helio Rangel

2

AULA 4

Criando o Projeto WEB no Visual Studio

Criando o Projeto WEB no Visual Studio

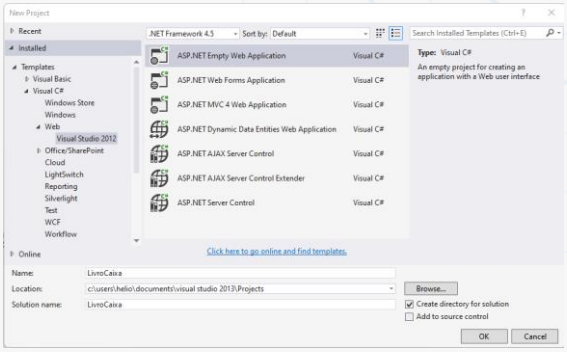
- O Visual Studio possui um leque variado de *templates* (modelos) preparados para facilitar a vida dos usuários e agilizar o desenvolvimento de aplicações de todos os tipos e, entre elas vários *templates* especializados em *WEB*.
- Apenas o estudo desses *templates* ocuparia bem mais que todo nosso espaço que temos aqui. Por isso, optamos por utilizar o mais básico dos *templates* para que possamos mostrar o mínimo necessário para criar uma aplicação completa *WEB***.**

Construindo o projeto

- Procurando fechar o leque de opções, criaremos nosso projeto utilizando o *template* mais básico a disposição e seguindo o caminho:

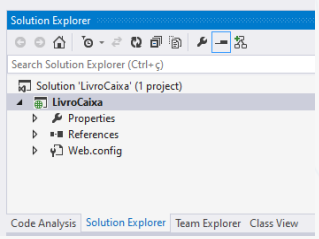
  FILE -> New -> Project ->

  Abrindo: Templates, Visual C#, Web, Visual Studio e selecionando o Template: *Empty Web Application*



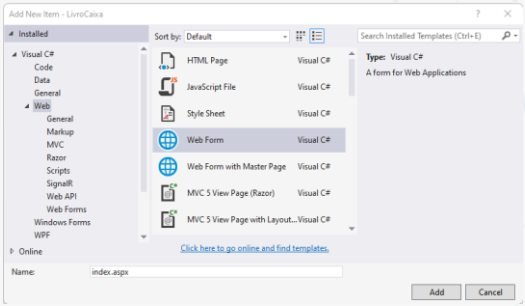
Construindo o projeto

- Ao clicar em OK, obtemos a seguinte tela na aba *Solution Explorer*:



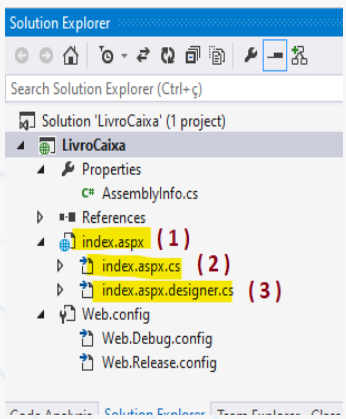
- Clicando com o botão direito sobre o nome do projeto (***LivroCaixa***), abrimos a tela:
- Seguindo o caminho: **Add -> New Item**

Construindo o projeto



- Selecionamos o tipo ***Web Form*** com o nome de **Index**.aspx.
- O sistema assume que, se existir um arquivo extensão html/aspx com este nome, ele será a página inicial do sistema.

Construindo o projeto



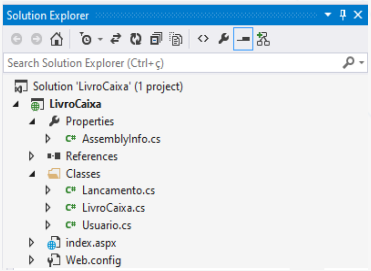
O Visual Studio, para cada Web Form adicionado cria um grupo de 3 arquivos, todos com o mesmo nome raiz, mas com extensões diferentes.

1. Apesar da extensão aspx, é um arquivo html onde vamos editar os componentes visuais necessários. É o nosso ***Front End***
2. Index.aspx.cs, é a classe Back End do formulário. Nele vamos editar nosso código C#
3. Index.aspx.Designer: Arquivo onde o Visual Studio define todos os objetos criados/editados no form

   (1). Este arquivo não deve ser editado diretamente pelo desenvolvedor.

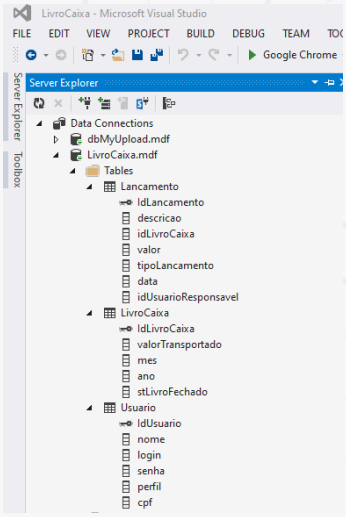
Criando as classes

- Com o botão direito sobre o nome do projeto (*Solution Explorer*) vamos codificar uma classe de cada vez, apesar do C# não exigir que cada classe seja codificada em um arquivo separadamente, recomenda-se criar uma pasta no projeto para manter as classes separadas do restante do projeto. Com isso o *Solution Explorer* fica:



Criando as tabelas

- Próximo passo seria criar as tabelas especificadas no projeto utilizando a aba *Server Explorer* do projeto.
- Por enquanto, isso é tudo que precisamos fazer no Banco de Dados.



A Classe de Conexão

Para que seja possível conectar com o Banco de Dados, precisamos construir uma classe com métodos para abrir/fechar conexões, tratar de transações, realizar consultas e alterações entre outros.

- O aluno deve examinar atentamente cada um desses métodos procurando compreender como funcionam e porque estão aqui.
- Na classe de conexão não devemos fazer referências diretas as classes de entidade, apenas disponibilizamos os serviços diretamente ligados as funcionalidades do Banco.
- Em sistemas *WEB* as conexões físicas com o Banco de Dados são abertas e liberadas para cada cliente que esteja acessando o sistema.
- Existem algumas situações de clientes acessando e alterando tabelas de forma "simultânea" e os serviços de inserção, alteração e consulta precisam dar conta do recado.
- Sempre que utilizamos um serviço do banco devemos tratar as exceções utilizando blocos ***try/catch*** para que seja possível lidar com os eventuais conflitos.

Note que ao abrir uma conexão, passamos o usuário como argumento. Fazemos isto para quando necessário, saber quem são os usuários responsáveis pelas transações no Banco de Dados.

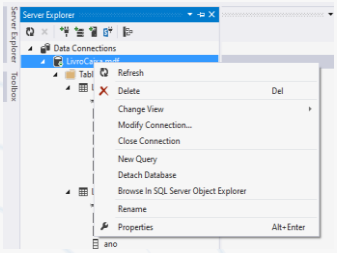
O pool de conexões

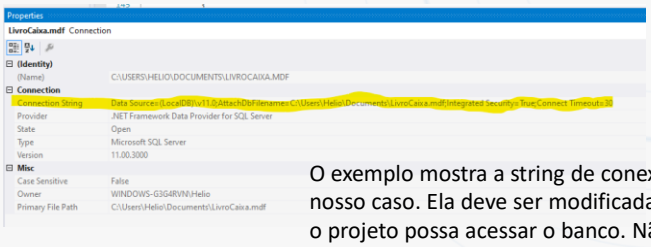
- Trabalhamos com conexões que são abertas e fechadas visando algumas transações específicas e, devemos manter estas conexões abertas o menor tempo possível para que outras transações, encontrem as tabelas já liberadas como forma de evitar colisões e conflitos que são os principais responsáveis por lentidão nas operações.
- Uma forma de fazer isso é utilizar mecanismos de fechamento automático como é o caso do bloco ***using*** que acessa o Banco fechando a conexão no momento que termina o bloco. A abertura da conexão deve ser explícita mas o fechamento é automático graças a implementação da interface *IDisposable* na classe Conexao.
- Exemplo:

```
using (Conexao con = new Conexao(usuario))
{
    con.open();
    usuario.atualiza(con);
    pnTrocaSenha.Visible = false;
    lbMensagem.Text = "Senha reiniciada com sucesso!";
}
```

A String de Conexões

- Quando criamos o banco de dados, o Visual Studio utiliza uma pasta local para criar o Banco. Esta localização é necessária para compor a *String de Conexão*. Clique com o botão direito sobre o nome do Banco e acesse a opção de Propriedades.






```
public static string stringConexao()
{
    return @"Data Source=(LocalDB)\v11.0;AttachDbFilename=C:\Users\Helio\Documents\LivroCaixa.mdf;" +
           "Integrated Security=True;Connect Timeout=30;Language=Portuguese";
}
```

O exemplo mostra a string de conexão para o nosso caso. Ela deve ser modificada para que o projeto possa acessar o banco. Não esqueça de acrescentar ao final da string ";Language=Portuguese". Esta é, basicamente, a única modificação necessária na classe Conexao para o sistema rodar.

Os códigos fonte

- Os códigos fonte das classes de conexão e classes entidade: Usuario, Lancamento e LivroCaixa, estão disponíveis na íntegra no anexo ao Guia de Disciplina.
- É muito importante que o aluno estude e entenda todas elas antes de prosseguir com os estudos