EAD UNISANTA

Programação Back End

Me. Helio A. L. Rangel

GUIA DA
DISCIPLINA

1. O que é a Internet

Objetivo:

Conceituar a Internet e seu funcionamento.

Introdução:

A Internet surgiu por volta dos anos 1960, na época da Guerra Fria, nos Estados Unidos. O Departamento de Defesa dos EUA planejou criar uma rede de computadores em pontos estratégicos do planeta. A ideia era descentralizar informações em vários servidores instalados em lugares geograficamente distantes de forma que não fossem destruídas no caso de um ataque nuclear por exemplo.

A ***ARPA*** (Advanced Research Projects Agency), uma das subdivisões do Departamento, implementou uma rede, a ***ARPANET***, conectada por um *backbone*, isto é, estruturas de rede capazes de manipular grandes volumes de informações. O acesso à ***ARPANET*** era restrito apenas a militares e pesquisadores, demorou um pouco até chegar ao público em geral, pois temiam o mau uso da rede pelo público em geral e países não aliados.

No Brasil, a conexão de computadores por uma rede somente era possível para fins estatais. Em 1991, a comunidade acadêmica brasileira conseguiu, através do Ministério da Ciência e Tecnologia, acesso a redes de pesquisas internacionais.

Em maio de 1995, a rede foi aberta ao público, ficando a cargo da iniciativa privada a exploração dos serviços.

Não demorou para o mundo dos negócios perceber o gigantesco potencial da ***INTERNET***, o custo muito reduzido se comparado as redes de computadores em uso na época, como era o caso de linhas telefônicas privadas, ou transmissão via rádio. Naturalmente, nem tudo são flores, ao mesmo tempo que a ***INTERNET*** colocava a disposição do grande público uma rede acessível e de baixo custo, mas que podia ser facilmente interceptada e os dados trafegados revelados sem grandes dificuldades. A criptografia, *firewalls* e outros dispositivos de segurança foram utilizados para minimizar as limitações de segurança da rede e permitir o seu uso coorporativo com relativa segurança.

1.1 Como funciona a internet?

Para que seja possível enviar uma mensagem para um computador, precisamos especificar qual é este computador. Por isso, qualquer computador conectado à uma rede deve possuir um endereço inequívoco de identificação, conhecido por "Endereço IP"

(Internet Protocol). Este é um endereço composto por uma série de 4 números entre 0 e 255, (0 e FF em hexadecimal) separados por pontos, por exemplo: 192.168.2.10.

Para nos ajudar a lembrar estes endereços, costumamos associar nomes aos endereços IP. São os chamados nome de domínio. Por exemplo, google.com é um nome de domínio usado para fazer referência o endereço 142.250.190.78. (Se digitar este número na linha de URL do navegador, deverá abrir a página do google)

O uso de 4 números para endereços de internet (IPv4), está sendo trocado para o chamado IPv6, que permite muitos bilhões de endereços no lugar dos antigos endereços possíveis com o IPv4 (4.294.967.295 combinações possíveis). O IPv4 parece um número gigantesco, mas é insuficiente para atender as necessidades de endereçamento modernas. Atualmente, ainda utilizando o IPv4, contornamos estas limitações utilizando o recurso do IP dinâmico, ou seja, nossos dispositivos conectados na internet possuem um endereço "provisório", enquanto estamos conectados. Este endereço, depois de utilizados, podem ser liberados e reutilizados em outras conexões de outros dispositivos.

1.2 Como os dados trafegam na INTERNET

Para que uma "conversa" entre duas pessoas ou "coisas" seja possível, precisamos definir regras claras de comportamento para que a "conversa" não seja extraviada, deturpada ou truncada. As regras que viabilizam esta conversa, chamamos de **Protocolo TCP/IP.** O Protocolo TCP/IP funciona como uma linguagem universal, que pode ser compreendida por computadores de qualquer fabricante, compatível com todos os Sistemas Operacionais.

Os dados que trafegam são embalados em "pacotes" de tamanho fixo com um "cabeçalho" contendo o número sequencial do pacote, endereço de destino além de outras informações como é claro, o conteúdo da informação propriamente dita. Os dados transmitidos através da Internet são agrupados em pacotes TCP (também é conhecido como datagrama) podem conter cada um até 1460 bytes de dados

A comutação de pacotes, é um conceito que pode ser atribuído a Paul Baran no início dos anos 1960 e bem mais tarde, de forma independente, a Donald Davies e Leonard Kleinrock.

Ao chegar ao destino, o pacote é armazenado o endereço de origem, e adicionada a aplicação na camada de Transporte, que realiza a ação pedida na camada da Aplicação, encapsula a resposta em outro pacote TCP/IP, coloca como destino o endereço de origem armazenado e insere seu endereço como o de origem.

Se, após um período pré-definido, o pacote esperado não tiver sido recebido, o TCP assume que o pacote foi perdido e solicita que o retransmita.

O roteador ao receber o pacote efetua a leitura da camada de Internet, verifica o endereço de destino, checa a lista interna de rotas que possui, e direciona o pacote para o caminho adequado, que pode ser o caminho mais longo com menor tráfego ou o mais curto.

Os pacotes enviados podem percorrer caminhos diferentes, desta forma não é raro que os pacotes sejam recebidos em ordem diferente da sua transmissão. Por isso os "cabeçalhos" dos pacotes carregam o número sequencial da transmissão para que o TCP/IP possa reordenar a os dados na ordem correta.

1.3 Os servidores de Internet

Um servidor de internet é um dispositivo físico, geralmente um computador, usado para monitorar e controlar os acessos da rede, sejam eles produzidos internamente ou recebidos de fora, via internet. Portanto, um servidor de internet é um equipamento cuja finalidade é viabilizar e controlar os acessos a rede. É por ele que todos os pacotes de dados passam antes de entrar/sair da Internet. Geralmente instalado na própria empresa com as funções de controlar as informações que trafegam, de forma que a organização possa, entre outras coisas, monitorar a navegação dos colaboradores. O servidor permite ainda adicionar funções de cache nas páginas, conversas em *chats* e ainda implementar funções de *firewall*.

1.4 Os navegadores de Internet.

Um navegador de Internet permite que o usuário possa acessar a qualquer endereço na Internet. Ele obtém informações de outras partes da Internet e apresenta-as no dispositivo onde estiver instalado. As informações são transferidas utilizando o Protocolo de Transferência de Hipertexto (Hyper Text Transfer Protocol - HTTP), que define como textos, imagens e vídeos trafegam. Estas informações precisam ser partilhadas e apresentadas num formato consistente para que as pessoas que utilizam um qualquer navegador, num qualquer lugar do mundo possam ver a informação.

Nem todos os navegadores interpretam o formato da mesma maneira. Para os usuários, isto significa que um site pode ter uma aparência e um funcionamento diferentes em navegadores diferentes.

Quando o navegador transfere dados de um servidor ligado à Internet, utiliza uma parte de *software* conhecido por mecanismo de renderização para traduzir estes dados em texto e imagens. Estes dados são formatados em **H**yper**T**ext **M**arkup **L**anguage (**HTML**) e

os navegadores de Internet interpretam este código para criar o que vemos, ouvimos e experimentamos na Internet.

Hiper ligações permitem que os usuários sigam um percurso para outras páginas ou sites na Internet. Cada página da Internet, imagem e/ou vídeo tem seu próprio **U**niform **R**esource **L**ocator (**URL**), que também é conhecido como endereço de Internet. Quando um navegador acessa um servidor em busca de dados, o endereço de Internet informa o navegador sobre onde procurar cada item descrito no **HTML**, que então indica ao navegador onde cada recurso deve ser colocado na página de Internet.

1.5 Os Cookies

Os sites podem armazenar informações em pastas conhecidos por *cookies*. Eles são especialmente úteis para armazenar dados relacionados com a navegação feita pelo usuário. Com o uso de cookies podemos “lembrar” de dados dos mais diversos criados durante navegações anteriores feitas no nosso site. O *Cookies* são armazenados com chaves de acesso que são unicamente acessáveis pela aplicação que os criou. Desta forma um *cookie* é praticamente inacessível por outras aplicações. Eles podem ser criados e acessados livremente pela aplicação *web back end* de forma prática e segura.

Os navegadores podem ser configurados para não aceitar *cookies*, mas, hoje em dia, praticamente todos os aplicativos web utilizam cookies e fica bastante complicado manter esta limitação nos navegadores.

Uma característica interessante dos *cookies* é que ao serem criados, o código pode determinar o tempo de duração deste *cookie* no cliente, des de alguns segundos até anos.

1.6 Aplicações WEB

As aplicações utilizando a internet são hoje uma tendência substituindo as antigas soluções desk top. O uso da internet como forma de viabilizar aplicações em rede trouxe uma certa complexidade a arquitetura de *software*, uma vez que é necessário a utilização de diversas tecnologias e soluções diferentes, combinadas e nem sempre 100% compatíveis entre si. Precisamos utilizar combinar tecnologias como HTML, Java Script, CSS, linguagens de programação como o Java, Ruby, Python, C#, PHP entre outras para que seja possível realizar a solução completa de *software*.

A segurança também é uma preocupação constante uma vez que, como vimos, a Internet não é por natureza, uma plataforma segura para tráfego de informações confidenciais;

Precisamos ressaltar aqui a importância de um projeto robusto da aplicação, desenvolvida e documentada, com todos os artefatos de projeto que servirão de base para o desenvolvimento da aplicação de forma a garantir o sucesso do projeto em respeito aos requisitos, prazo e orçamento.

O projeto da base de dados também não deve ser de forma alguma subestimado. Uma base de dados desenvolvida utilizando-se as melhores práticas de modelagem e normlaização também é fundamental para o sucesso da empreitada;

1.7 Algumas tecnologias de desenvolvimento WEB

1.7.1 Visual Studio / ASP.NET

O asp.net é uma plataforma completa de desenvolvimento de software, inclusive WEB, colocada no mercado pela Microsoft. É possível desenvolver aplicações em diversas linguagens, mas a que mais se recomenda é o C#, uma linguagem sofisticada que também foi desenvolvida pela Microsoft e que é 100% Orientada a Objetos.

1.7.2 Java

Java é uma linguagem de programação orientada a objetos que é capaz de desenvolver *softwares* que rodam em diferentes plataformas sem que seja necessário modificá-los. É uma derivação de C e C++, tecnologias muito potentes, porém um tanto complexas,

1.7.3 Javascript

É uma linguagem de script que roda no cliente. Mudou a maneira como os sites eram desenvolvidos e utilizados. Trouxe mais dinamismo e interatividade para as páginas da internet. É possível encontrar, através de bibliotecas, aplicações JavaScript até mesmo **Back End**.

1.7.4 Ruby

O Ruby é uma linguagem de script interpretada e multiplataforma. Ganhou muita popularidade nos últimos anos após a criação do framework Ruby on Rails, voltado para a criação de sites. Por ser muito similar ao Python, linguagem cuja acessibilidade é um de seus maiores atrativos, o Ruby também possui uma curva de aprendizagem bastante interessante.

1.7.5 PHP

O PHP também é uma linguagem de script, de código aberto, voltada à programação **Back End**. Os códigos PHP são executados em um servidor, e não diretamente no

navegador. É muito indicada para iniciantes em programação porque é bastante simples, ao mesmo tempo em que possui uma longa lista de funções bastante profissionais.

1.7.6 Python

Criada para otimizar a criação de códigos e estimular a produtividade dos programadores, esta linguagem, além de super completa e abrangente, possui uma sintaxe extremamente compreensível. Muito utilizada nas áreas de inteligência artificial, machine learning e ciência de dados, o Python disponibiliza um grande número de bibliotecas de extrema utilidade.

2. Desenvolvimento Back End

Objetivo:

Entender o que se considera como programação Back End, qual o mínimo necessário para se iniciar como desenvolvedor e conhecer técnicas básicas de programação.

Introdução:

Programar em qualquer linguagem, exige um raciocínio lógico matemático que nem sempre é fácil para algumas pessoas. Mas o estudo das técnicas e a prática do dia a dia na programação acaba nos conferindo o que é necessário para se ter sucesso.

O *Back End* é a parte do sistema responsável pela lógica de negócios, persistência, controle de permissões de acesso, consistência de dados e informações, segurança e uma série de coisas que podemos considerar como o “organismo” do sistema. O Front End pode ser comparado a nossa aparência, voz, pele, olhos, olfato, órgãos dos sentidos, enquanto o Back End seria o cérebro, coração, fígado, rins etc. Como em nosso organismo, para que o conjunto funcione perfeitamente a comunicação entre o Front End e o Back End precisa ser perfeita.

2.1 A Arquitetura MVC

A arquitetura MVC (Model, View, Controller) tem sido largamente utilizada e consagrada como forma mais adequada, segura e produtiva de se desenvolver aplicações e sistemas principalmente WEB. Existem algumas variações na maneira como adotamos esta arquitetura mas basicamente ela é entendida como:

2.1.1 Model

A representação "domínio" específica da informação em que a aplicação opera. Lógica de domínio adiciona sentido a dados. Muitas aplicações usam um mecanismo de armazenamento persistente (como banco de dados) para armazenar dados. MVC não cita especificamente a camada para acesso aos dados, porque subentende-se que estes métodos estariam encapsulados pelo Model.

2.1.2 View

Renderiza o model em uma forma específica para a interação, geralmente uma interface de usuário.

2.1.3 Controller

Processa e responde a eventos, geralmente ações do usuário, e pode invocar alterações no Model. É lá que é feita a validação dos dados e também é onde os valores postos pelos usuários são filtrados.

2.2 As linguagens mais utilizadas

2.2.1 PHP (Hypertext Preprocessor)

É uma linguagem de script *open source* de uso geral, muito utilizada, e especialmente pensada para o desenvolvimento web e que pode ser editada dentro da página HTML. O PHP permite uma certa mistura nos conceitos de Back End e Front End. É uma das linguagens WEB mais utilizadas e produtivas. Existem *frameworks* disponíveis que permitem desenvolver em PHP utilizando conceitos de Orientação a Objetos e que viabilizam uma separação mais clara entre os projetos de Front End e Back End (MVC).

O que distingue o PHP de algo como o Javascript no lado do cliente é que o código é executado no servidor, gerando a página HTML que é então enviada para o navegador. O navegador recebe os resultados da execução desse script, mas não conhece o código fonte que deu origem a página. Um ponto forte do PHP é que ele é considerado simples para um iniciante, e mesmo assim, oferece recursos bastante avançados para um desenvolvedor profissional.

O código PHP normalmente é escrito diretamente na página HTML

Exemplo:

```
    <!DOCTYPE HTML>
<html>
    <head>
        <title>Exemplo</title>
    </head>
    <body>
        <?php echo "Oi geeeenti!"; ?>
    </body>
</html>
```

2.2.2 Java:

Uma das primeiras linguagens a ser utilizada na WEB através dos Applets (que rodam no navegador) e Servlets (rodam no servidor). Conhecer os servlets permite não

somente entender melhor esses frameworks, como também fornece ao desenvolvedor uma capacidade maior de interação com eles.

2.2.2.1 Sistema cliente e servidor

Quando falamos em aplicações web, estamos nos referindo a sistemas ou sites onde grande parte da programação fica hospedada em servidores na internet, e o usuário normalmente não precisa ter nada instalado em sua máquina para utilizá-las. Na plataforma Java, esse modelo foi implementado através da Interface de Programação de Aplicação (API) de *Servlets*. Um *Servlet* estende a funcionalidade de um servidor web para servir páginas dinâmicas aos navegadores, utilizando o protocolo HTTP.

No mundo Java, os servidores web são chamados de *Servlet Container*, pois implementam a especificação de *Servlet*. O servidor converte a requisição em um objeto do tipo HttpServletRequest. Este objeto é então passado aos componentes web, que podem executar qualquer código Java para que possa ser gerado um conteúdo dinâmico. Em seguida, o componente web devolve um objeto HttpServletResponse, que representa a resposta ao cliente. Este objeto é utilizado para que o conteúdo gerado seja enviado ao navegador do usuário.

2.2.2.2 Servidores de aplicação WEB (Containers)

São interfaces entre componentes. Para que uma aplicação web desenvolvida em Java ou um componente corporativo possa ser executado, eles precisam ser implantados em um servidor de aplicação. Existem alguns servidores disponíveis no mercado. O EJB Container, por exemplo, suporta *Enterprise JavaBeans* (EJB), que são componentes corporativos distribuídos. Os *Servlets*, JSP (*Java Server Pages*), páginas JSF (*Java Server Faces*) e arquivos estáticos (HTML, CSS, imagens etc.) necessitam de um servidor de aplicação para ser executado.

Um conhecido exemplo de container (servidor de aplicações) é o Apache Tomcat, que é leve, gratuito e muito popular.

O JSP é um *framework* de desenvolvimento web em Java que permite desenvolver aplicações rápidas e enxutas com uma interface *Back End/Front End* baseada na notação *JavaBeans*. Esta notação se resume a criar métodos escritos de forma peculiar que podem ser chamados diretamente das páginas JSP/JSF.

Pesquisar

- O que é JavaBeans;
- Quais são as notações utilizadas;
- Para exatamente o que serve um servidor de aplicação?
- Aplicações MVC o que são e quais vantagens.

2.2.3 O ASP.NET

A plataforma ASP.NET é uma solução integrada proprietária Microsoft que permite o desenvolvimento de aplicações WEB utilizando linguagens como C#, VB.NET, F# entre outras. A linguagem atualmente mais em uso na plataforma .NET é o C# (*C Sharp*). O C# é uma linguagem 100% orientada a objetos com algumas semelhanças com C++ e Java. Por ser um "pacote" de desenvolvimento, oferece uma série de possibilidades e "*templates*" para sistemas WEB com as mais diversas aplicações. Para quem conhece Java ou C++, o aprendizado do C# passa a ser bastante intuitivo e sem grandes surpresas.

O desenvolvimento de sistemas em ASP.NET é uma experiência intuitiva e com uma excelente curva de aprendizagem, veloz e produtiva. Um ponto considerado fraco na plataforma ASP.NET é o editor de páginas para uso dos desenvolvedores de ***Front End***. Contudo, os desenvolvedores ***Back End*** não terão do que se queixar.

2.2.4 Persistência

Normalmente a persistência dos dados fica a cargo do ***Back End***. O mais comum é utilizar um banco de dados relacional de mercado. Bancos como MySql, Oracle, SqlServer, PostgreSQL entre outros são os mais utilizados e sua escolha está associada a razões econômicas e/ou técnicas. Bancos especiais do tipo NoSQL ou serialização de objetos também não são incomuns. O volume de dados, a necessidade de consultas e atualizações rápidas, a segurança, confiabilidade, praticidade, custo, escalabilidade e disponibilidade são alguns dos fatores levados em conta no momento da decisão de qual mecanismo de persistência será adotado.

A arquitetura MVC recomenda uma separação clara entre a persistência e as demais camadas do sistema. Esta separação ajuda bastante o desenvolvedor ***Back End*** a pensar separadamente nos processos de persistência, como comandos SQL, conexões etc., e as regras de negócio.

Em linguagens OO, recomenda-se que cada tabela identidade possua sua própria classe de controle onde podemos codificar tudo que for necessário para persistência dos dados de forma que as classes back end dos formulários fiquem livres de código SQL.

2.2.5 Stored Procedures

É uma conjunto de comandos em SQL que podem ser executados de uma só vez, como em uma função. Ele armazena tarefas repetitivas e aceita parâmetros de entrada para que a tarefa seja efetuada de acordo com a necessidade individual.

Um Stored Procedure pode reduzir o tráfego na rede, melhorar a performance de um banco de dados, criar tarefas agendadas, diminuir riscos, criar rotinas de processamento etc.

2.2.5.1 Quando se recomenda

Quando temos várias aplicações escritas em diferentes linguagens, ou rodam em plataformas diferentes, porém executam a mesma função.

Quando damos prioridade à consistência e segurança.

O desempenho é um item crítico.

2.2.5.2 Desvantagens

Nas Stored Procedures acabamos implementando regras de negócio. Desta forma ficamos mais dependentes dos bancos de dados e diminuímos a independência das camadas MVC.

A migração de uma tecnologia de persistência para outra pode ficar mais trabalhosa, cara e complexa.

Em algumas organizações o time de ***back end*** não pode desenvolver e manter as *stored procedures* pois são de responsabilidade dos DBA's o que acaba trazendo alguns transtornos e dificuldades de comunicação entre as equipes do projeto.

3. O projeto do sistema

Objetivo:

Apresentar o escopo do projeto.

Introdução:

Muitas vezes subestimado por desenvolvedores iniciantes, o projeto é fundamental para que o produto atenda as expectativas do cliente. O levantamento de requisitos, prototipação, projeto de classes, modelagem do banco de dados são todos fundamentais para o sucesso final do projeto.

A melhor maneira de aprender programação é programando... e para isso precisamos de um projeto real, de preferência com aplicação e utilidade prática. Também vamos adotar o paradigma da Orientação a Objetos que oferece excelentes recursos e benefícios que facilitam a vida dos desenvolvedores, tornando o código mais robusto, confiável e manutenível.

A linguagem de programação que adotaremos é o C#, linguagem 100% orientada a objetos e que já foi objeto de estudos neste curso. Para o banco de dados, propomos o SGBD nativo do Visual Studio, o Sqlserver, como nossa melhor opção.

Infelizmente, a maneira que as várias tecnologias de desenvolvimento de aplicativos WEB encontraram para solucionar os desafios de arquitetura, são suficientemente diferentes para merecer estudos separados. Contudo, o conceito por traz continua o mesmo e o desenvolvedor, uma vez dominando um ambiente de desenvolvimento WEB, não deverá ter muita dificuldade para aprender outros.

3.1 O escopo do projeto

Vamos desenvolver um sistema de **Livro Caixa**. Apesar do foco ser o ***Back End***, não tem como desenvolver um sistema apenas de ***Back End***. Assim, vamos desenvolver uma interface (***Front End***) extremamente simples, o mínimo necessário para realizar o nosso projeto. Fica a critério do aluno sofisticar e/ou melhorar a interface dependendo, é claro, de seus conhecimentos no mundo do ***Front End***.

O Sistema propõe substituir o clássico livro caixa usados para controlar entradas e saídas de recursos para controlar receitas e despesas de uma instituição qualquer. Uma simples planilha Excel pode fazer este serviço de forma eficiente. Todavia, requisitos como controle de acesso, acesso remoto, portabilidade dos dados (via BD), não precisar ter o *Excel* instalado no dispositivo do cliente entre outros, pode justificar a criação de um sistema

WEB específico para este fim, uma vez que tentaram implementar esses requisitos no *Excel* não é uma tarefa trivial.

Os artefatos utilizados para definir o projeto estão simplificados e tem o objetivo de deixar clara a especificação da solução de *software* a ser desenvolvida.

3.1.1 Os Requisitos

3.1.1.1 Não Funcionais

a) Sistema para WEB
b) Desenvolvido na plataforma .NET em C#
c) Utilizando Banco de Dados Sqlserver (Nativo do ambiente)
d) Com controle de acesso (usuário e senha).
e) A tela de abertura (login) deve possuir o título: LIVRO CAIXA – XXXXXX, onde XXXXX é o nome da empresa. O Nome da empresa deve ser armazenado como cookie e informado no primeiro acesso (caso o cookie não exista, ou a qualquer momento quando o usuário clicar no nome da empresa.
f) O nome da empresa deve aparecer também no cabeçalho das tabelas de usuários e lançamentos
g) Na tela de Lançamentos e Cadastro de Usuários, mostrar no canto superior direto o nome do usuário registrado.

3.1.1.2 Funcionais (Login e Menu)

a) A tela inicial de login deverá ter um campo para login, outro para senha e um botão de confirmar.
b) Caso o usuário registrado seja ADM, abre a tela de menu com duas opções (implementadas como links): CAD USUÁRIO e LIVRO CAIXA.
c) Caso o usuário não seja ADM, não mostra as opções do menu abrindo direto a tela de fluxo de caixa.
d) Se o usuário registrado digitar o seu próprio CPF, a tela passará a mostrar dois campos de senha (o segundo é campo de confirmação da senha) que irá permitir que o usuário altere sua senha inicial.

3.1.1.3 Funcionais (Cadastro de usuários)

a) A opção no menu de cadastro de usuários aparece apenas para usuários perfil ADM.
b) Os usuários devem possuir os perfis: ADM, podem cadastrar usuários e operar todas as funções do sistema. USU: Apenas operar o sistema de Lançamentos;

c) Cadastro de usuários: Campos: Login, Senha, Nome Completo, CPF, Perfil (Radio Button: ADM e USU) – Todos os campos são obrigatórios.)
d) Senha com no mínimo 6 (seis), no máximo 10 (dez) dígitos (Sem restrições de dígitos / tipos)
e) A senha inicial é o CPF do usuário que deve ser trocada no primeiro acesso;
f) Permitir busca por sequencial / nome / login, edição (alteração e exclusão)
g) Mostrar os usuários cadastrados em uma tabela por ordem alfabética de nome;

3.1.1.4 Funcionais (Livro Caixa)

a) Ao abrir a tela de fluxo de caixa, o usuário escolhe o período (mês/ano do lançamento – utilizando dois combos, um de mês, outro de ano – O mês/ano corrente deve estar pré-selecionado)
b) Campos: Valor do lançamento, Descrição, Data/Hora, Tipo (débito ou crédito: Implementado em Radio Button), usuário registrado; O botão de Lançar deverá permanecer desabilitado no caso de livro fechado;
c) Tabela com todos os lançamentos, com todos os campos do item a), mais uma coluna com o saldo em caixa após cada lançamento, por ordem de Data/Hora;
d) Não existe saldo negativo. Caso o lançamento de débito seja superior ao saldo, o sistema não realiza o movimento mostrando mensagem na tela.
e) Nenhum lançamento pode ser editado ou apagado, caso necessário deve ser feitos lançamentos de estorno e/ou correção devidamente documentados na descrição do lançamento;
f) Implementar um botão de fechamento do período. Ao fechar o período o sistema fecha o período atual, cria um registro de LivroCaixa com o período seguinte, transporta o saldo para o período seguinte; este botão deverá ter um pedido de confirmação avisando que uma vez fechado, não poderá haver mais lançamentos; ele deve ser desabilitado caso o período já esteja fechado;
g) O livro fechado pode ser consultado, mas não aceitará mais lançamentos.

3.1.2 As classes entidade envolvidas

3.1.2.1 Usuário

Usuario
-idUsuario: int
-nome:String
-login:String
-senha:String
-perfil:String
-cpf: String

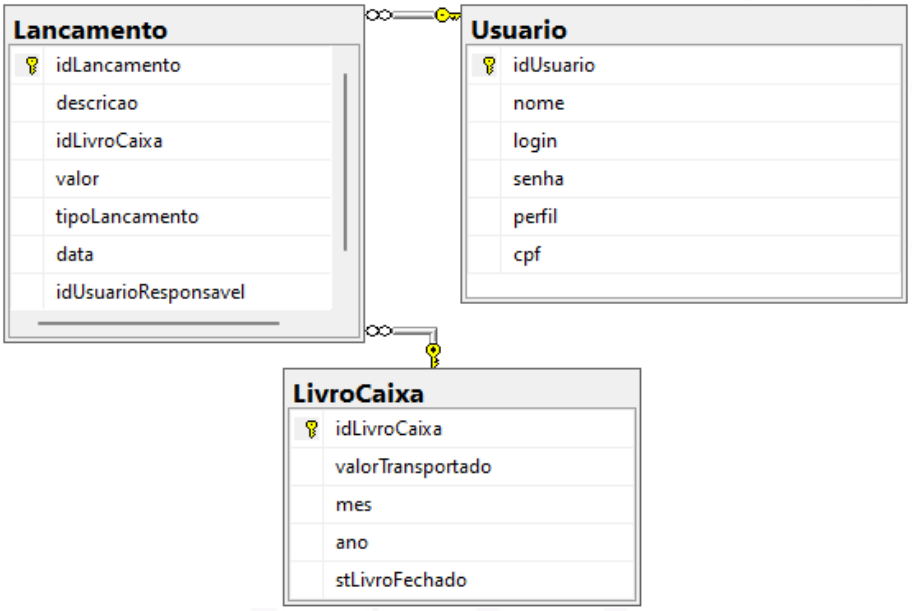
3.1.2.2 LivroCaixa

LivroCaixa
-idLivroCaixa: int
-valorTransportado:Double
-mes:int
-ano:int
-stLivroFechado

3.1.2.3 Lançamento

Lancamento
-idLancamento: int
-descricao:String
-idLivroCaixa: int
-valor:Double
-tipoLancamento: char
-data:DateTime
-idUsuarioResponsavel

3.1.3 Nosso MER



Obs.: Todas as tabelas são `IDENTITY (1,1)`

4. Criando o projeto WEB no Visual Studio

Objetivo:

Mostrar como construir o projeto na plataforma Visual Studio/asp.net e introduzir princípios de operação de Bancos de Dados em sistemas WEB

Introdução:

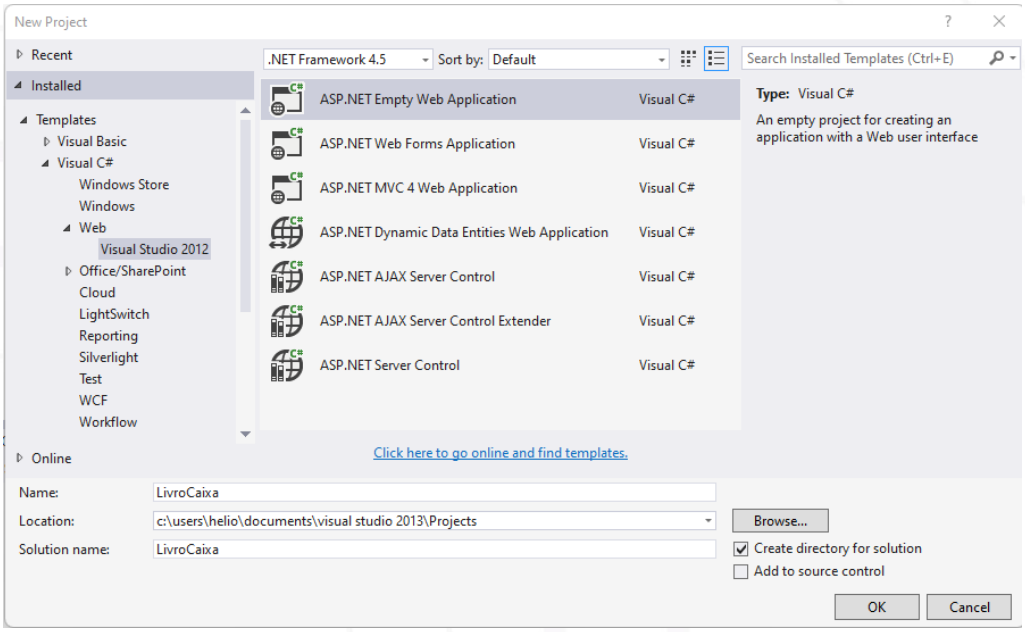
O Visual Studio possui um leque variado de *templates* preparados para facilitar a vida dos usuários e agilizar o desenvolvimento de aplicações de todos os tipos e, entre elas vários templates especializados em WEB. Apenas o estudo desses templates ocuparia bem mais que todo nosso espaço que temos aqui.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

4.1 Construindo o projeto

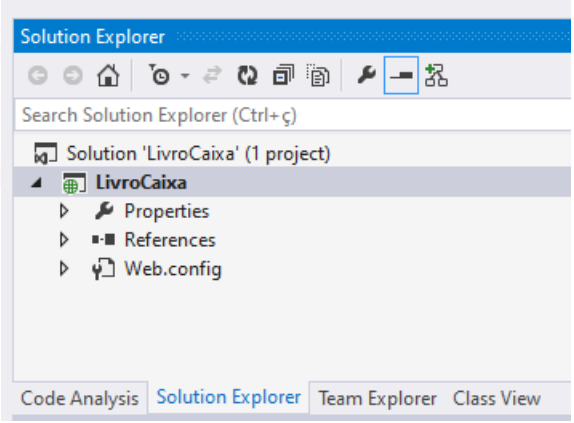
Como dissemos, o Visual Studio é um ambiente completo para gerar aplicações, de várias linguagens e plataformas. Procurando fechar o leque de opções, vamos criar nosso projeto utilizando o *template* mais básico a disposição seguindo o caminho:

`FILE -> New -> Project ->`

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

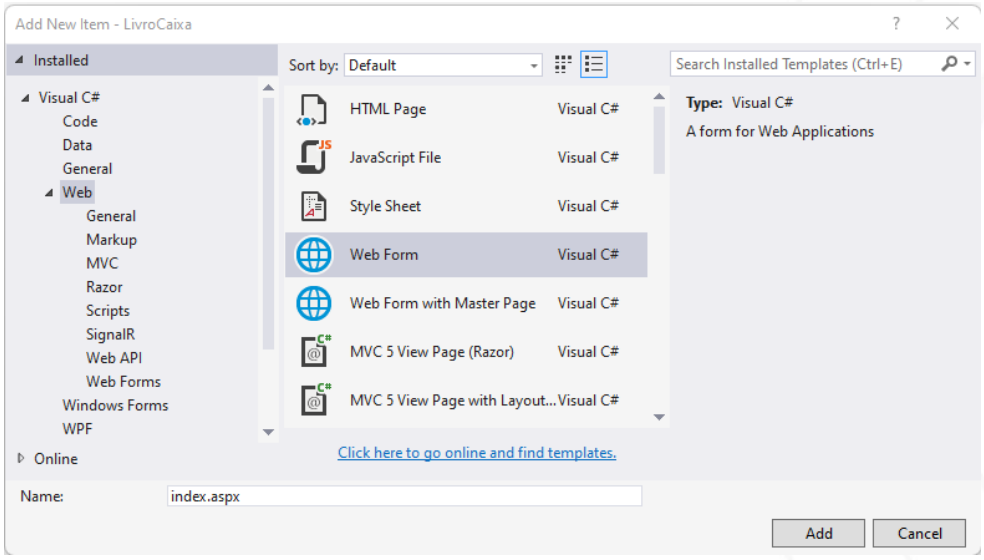


Ao clicar em OK, vamos obter a seguinte tela na aba *Solution Explorer*:

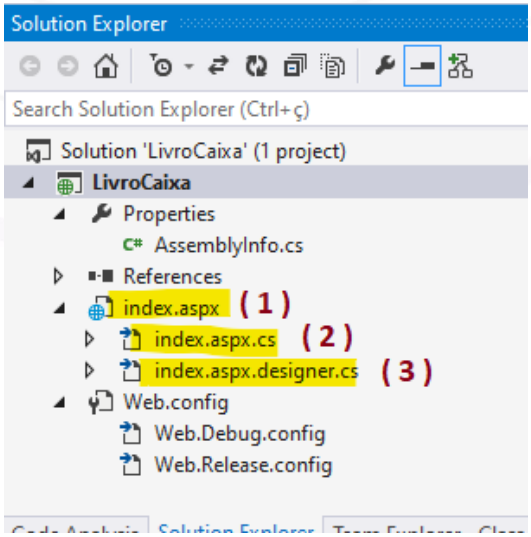


Clicando com o botão direito sobre o nome do projeto (***LivroCaixa***), abrimos a tela:

Seguindo o caminho: **`Add -> New Item`**



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.



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.

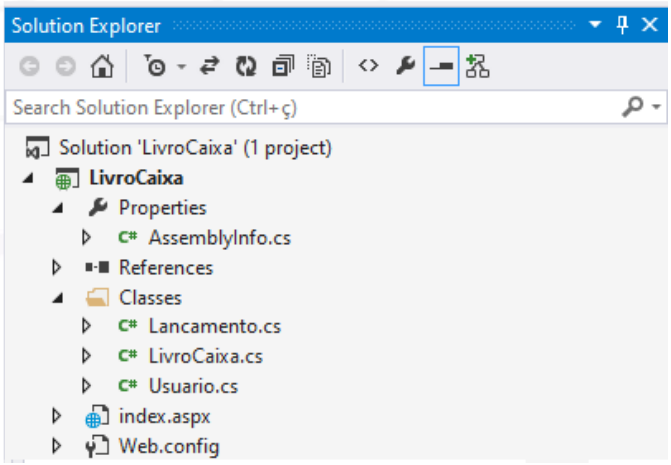
Entendendo

Antes de editar qualquer coisa nos arquivos do Form Index, abra cada um e examine procurando compreender como eles se relacionam entre si e funcionam.

Antes de iniciar a codificação dos forms, vamos criar cada uma das classes que levantamos na fase de análise:

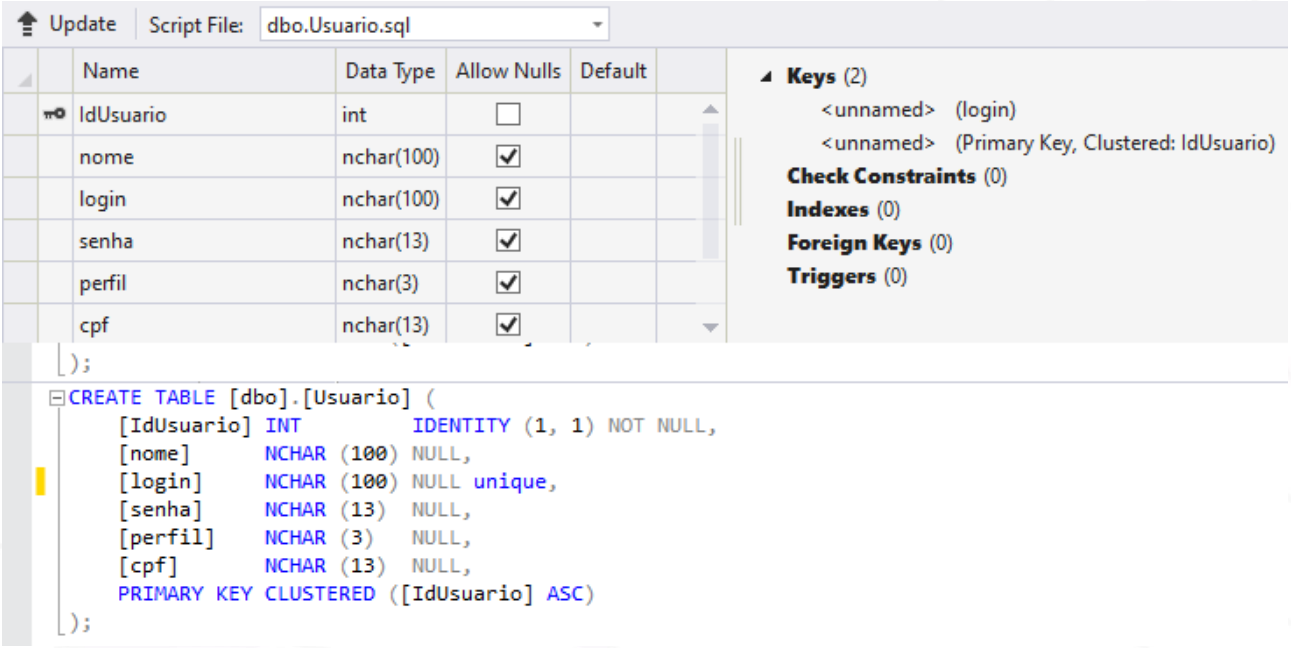
4.1.1 Criando as classes

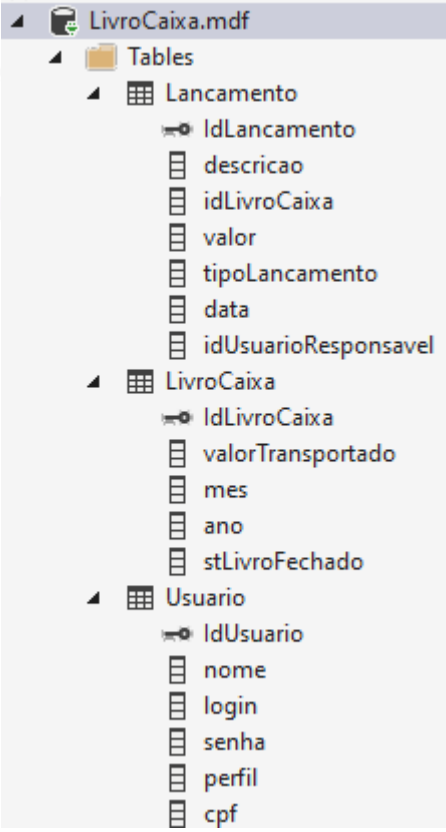
Com o botão direito sobre o nome do projeto (*Solution Explorer*) vamos codificar uma classe de cada vez, apesar do C# não exigir, como o Java, que cada classe seja codificada em um arquivo separadamente, recomenda-se criar uma pasta no projeto para manter as classes separadas do restante do projeto.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



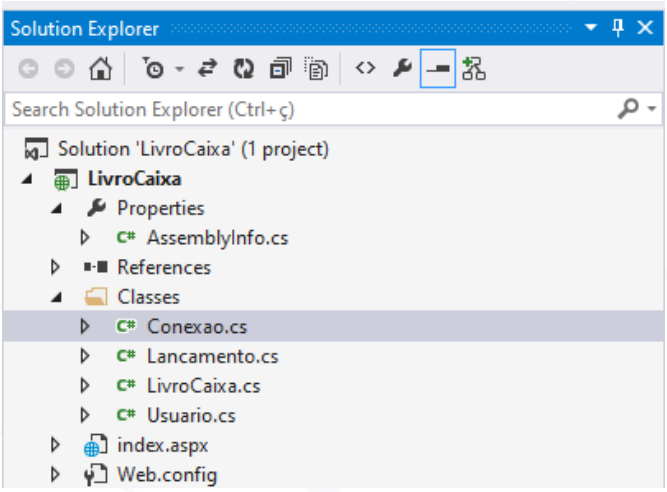
4.1.2 Criando as tabelas

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





Vamos criar a classe de conexão com o banco de dados:



4.1.3 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. Nela 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 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. O desenvolvedor pode contar que vai dar tudo certo, mas precisa saber lidar com os "erros" que podem, e certamente vão aparecer durante a operação. 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.

4.1.3.1 O pool de conexões

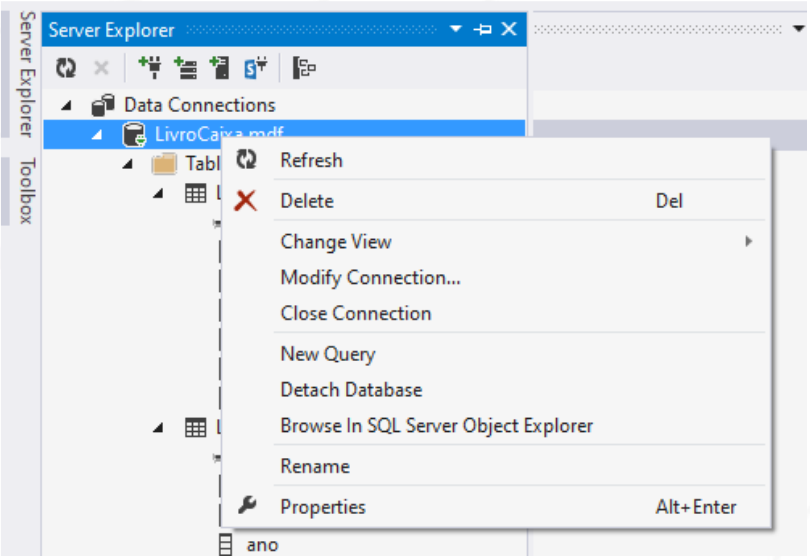
Como vimos, trabalhamos com conexões que são abertas e fechadas visando algumas transações específicas e, devemos manter estas conexões abertas o menor tempo possível para que outras transações, encontrem as tabelas já liberadas, evitando colisões e conflitos que são os principais responsáveis por lentidão nas operações. Uma forma de fazer isso é utilizar mecanismos de fechamento automático como é o caso do bloco ***using*** que acessa o banco de dados, 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.

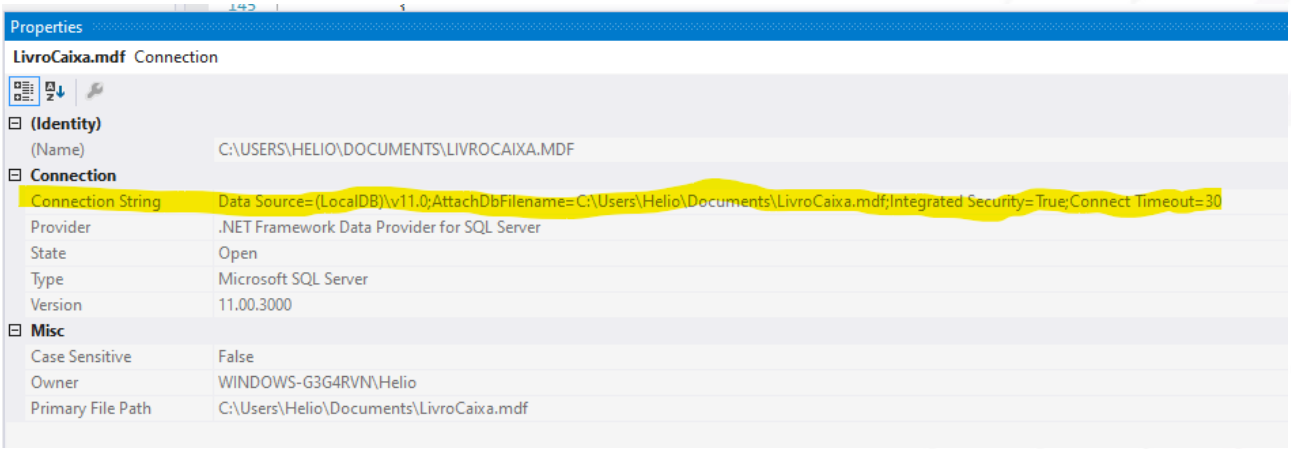
Exemplo:

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

4.1.3.2 A String de Conexão

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

4.1.4 As Classes Identidade

As classes identidade, são responsáveis pela interface com as tabelas do banco de dados de mesmo nome. Para cada atributo das tabelas, existe um atributo correspondente declarado na classe. Este recurso é particularmente interessante porque viabiliza a separação das camadas de persistência e modelo facilitando o acesso aos dados armazenados no banco de forma transparente para o usuário das classes. Nas classes identidade, também são codificados métodos para consulta, alteração, inserção e deleção

de registros e ainda alguns métodos que necessitam acesso ao banco de dados como busca de registros, montagem de listas, construção de objetos etc.

Saiba mais

Existem alguns *frameworks* de mercado que facilitam bastante esta tarefa de persistência de dados. Cada um deles merece um mini curso para se conhecer e utilizar corretamente. Aqui estamos mantendo o mesmo conceito e mostrando como funciona a ideia de forma bem simples. Pesquise e procure conhecer os *frameworks* de persistência de dados de mercado.

4.1.4.1 Buscando registros no banco de dados

```
  private static string nomeTabela = "Usuario";
  private static string nomeId = "id" + nomeTabela;
  private static string campos = "nome, login, senha, perfil, cpf";

public static Usuario busca(int id, Conexao con)
{
    try
    {
        String sql = String.Concat("SELECT ", campos, " FROM ", nomeTabela, " WHERE ",
            nomeId, "=", id);
        DataTable dt = Conexao.executaSelect(con, sql,null);
        if (dt.Rows.Count == 0) return null;
        DataRow[] r = dt.Select();
        Usuario item = new Usuario(r[0][0].ToString(), r[0][1].ToString(),
                            r[0][2].ToString(), r[0][3].ToString(), r[0][4].ToString());
        item.idUsuario = id;
        return item;
    }
    catch (Exception)
    {
        throw;
    }
}
```

O método busca(), recebe como argumento o ID (chave primária) do registro e a conexão já aberta. A busca do registro é feita montando-se uma query utilizando atributos como ‘campos’, ‘nomeTabela’ e ‘nomeId’ definidos no início de cada classe identidade. Cada uma delas inicializa essas variáveis com os nomes respectivos de cada tabela.

Exemplo no caso da classe LivroCaixa:

```
private static string nomeTabela = "LivroCaixa";
private static string nomeId = "id" + nomeTabela;
private static string campos = "valorTransportado, mes, ano, stLivroFechado";
```

Caso a consulta retorne o registro com sucesso, um objeto é instanciado e populado com todos os atributos carregados do banco que é finalmente retornado finalizando o

método. O processo de busca de registros na base de dados é o mesmo para cada uma das classes identidade.

4.1.4.2 Inserindo Registros

```
1 reference
public int insere(Conexao con)
{
    try
    {
        String[] args = { nome, login, senha, perfil, cpf };
        String sql = String.Concat("INSERT INTO ", nomeTabela,
            " (", campos, ") Values (@1,@2,@3,@4,@5)");
        idUsuario = Conexao.executaQuery(con, sql, nomeTabela,args);
        return idUsuario;
    }
    catch (Exception)
    {
        throw;
    }
}
```

Os métodos insere() são chamados via objeto com os atributos já inicializados. A conexão passada também deve estar aberta e, se for o caso, fazendo parte de uma transação.

```
using (Conexao con = new Conexao(usuario))
{
    con.open();
    Usuario u = Usuario.busca(con, txLogin.Text);
    if (u != null && u.idUsuario > 0)
    {
        lbMensagem.Text = "Já existe um usuário com este login: " + txLogin.Text;
        return;
    }
    txCPF.Text = txCPF.Text.Replace(".", "").Replace("/", "");
    u = new Usuario(txNome.Text, txLogin.Text, txCPF.Text, rbAdm.Checked ? "ADM" : "USU", txCPF.Text);
    u.insere(con);
    limpa();
    lbMensagem.Text = "Novo usuário cadastrado com sucesso!";
    relatorioUsuarios.Text = usuario.montaTabela((String)Session["empresa"], con);
}
```

O trecho de código acima prepara o objeto usuário para ser persistido (salvo no banco). Depois de verificar que não existe outro usuário com o mesmo login, o objeto usuário é instanciado e o método insere é chamado passando a conexão como argumento.

É possível garantir que não existe nenhum usuário de login idêntico, alterando o campo *login* da tabela *Usuario*, setando como ‘unique’

```
[login]     NCHAR (100) NULL unique,
```

4.1.4.3 Atualizando Registros

```
public int atualiza(Conexao con)
{
    try
    {
        string[] args = { nome, login, senha, perfil, cpf };

        StringBuilder sql = new StringBuilder("UPDATE " + nomeTabela + " SET ");
        sql.Append("nome=@1,");
        sql.Append("login=@2,");
        sql.Append("senha=@3,");
        sql.Append("perfil=@4,");
        sql.Append("cpf=@5 ");
        sql.Append(" WHERE id" + nomeTabela + "=" + idUsuario);

        Conexao.executaQuery(con, sql.ToString(), nomeTabela,args);

        return idUsuario;
    }
    catch (Exception)
    {
        throw;
    }
}
```

O método atualiza() utiliza preparação de queries assim como o método insere mostrado no item anterior. O método Conexao.exeutaQuery() recebe como argumento a conexão, a querie, o nome da tabela que esta sendo acessado e a lista de argumentos.

4.1.4.4 Excluindo Registros

```
public int exclui(Conexao con)
{
    try
    {
        String sql = String.Concat("DELETE FROM ", nomeTabela,
            " WHERE idUsuario=" , idUsuario);
        return Conexao.executaQuery(con, sql, nomeTabela,null);
    }
    catch (Exception)
    {
        throw;
    }
}
```

Importante

Examine com cuidado os métodos Conexao.executaQuery() e Conexao.executaSelect() tendo certeza que foram compreendidas.

Procure também conhecer melhor o funcionamento de exceções e transações de banco de dados.

5. A interface com o usuário (Front End)

Objetivo:

Mostrar a interface com o usuário desenvolvida, os protótipos das telas, discutindo brevemente os elementos gráficos utilizados em cada uma;

Introdução:

Apesar de não ser o ponto central do curso, precisamos desenvolver uma interface mínima para viabilizar um projeto WEB. Não pretendemos discutir nem detalhar aspectos de *design* mas não podemos deixar de discutir como os objetos gráficos são acessados além de como utilizar funções Javascript e um mínimo de CSS. O aluno não deverá considerar as soluções de *design* adotadas como soluções de boas práticas. Foram utilizadas por simplicidade para poder manter o foco no *Back End* que o nosso principal objetivo.

5.1 Os objetos do Front End que são acessíveis pelo Back End

O *ASP.Net* define componentes com a seguinte estrutura básica: Um Label por exemplo:

```
<asp:Label ID="lbUsuarioLogado" runat="server" Text="Usuário"></asp:Label>
```

A arquitetura da definição de objetos ASP, segue a ideia básica do HTML de criar tags que abrem e fecham. Estes objetos, naturalmente, podem ter suas propriedades modificadas na acamada do *Back End*, portanto na classe *CodeBehind* do Form Desmembrando as partes:

*5.1.1 O **<asp:***

Prefixo que indica que se trata de um objeto asp.net

*5.1.2 **Label***

Classe do objeto

*5.1.3 Propriedade: **ID="lbUsuarioLogado"***

Identificador do objeto

*5.1.4 Propriedade: **runat="server"***

Significa que o objeto roda no “server”, não é um objeto HTML comum. Todos os objetos asp.net rodam no servidor mas esta especificação existe abrir possibilidades futuras em novas versões do asp.net

5.1.5 Propriedade: ***Text="Usuário"***

A propriedade text assume o valor, neste caso, de ***Usuário***. Esta propriedade pode ser acessada na classe *CodeBehind* como no exemplo:

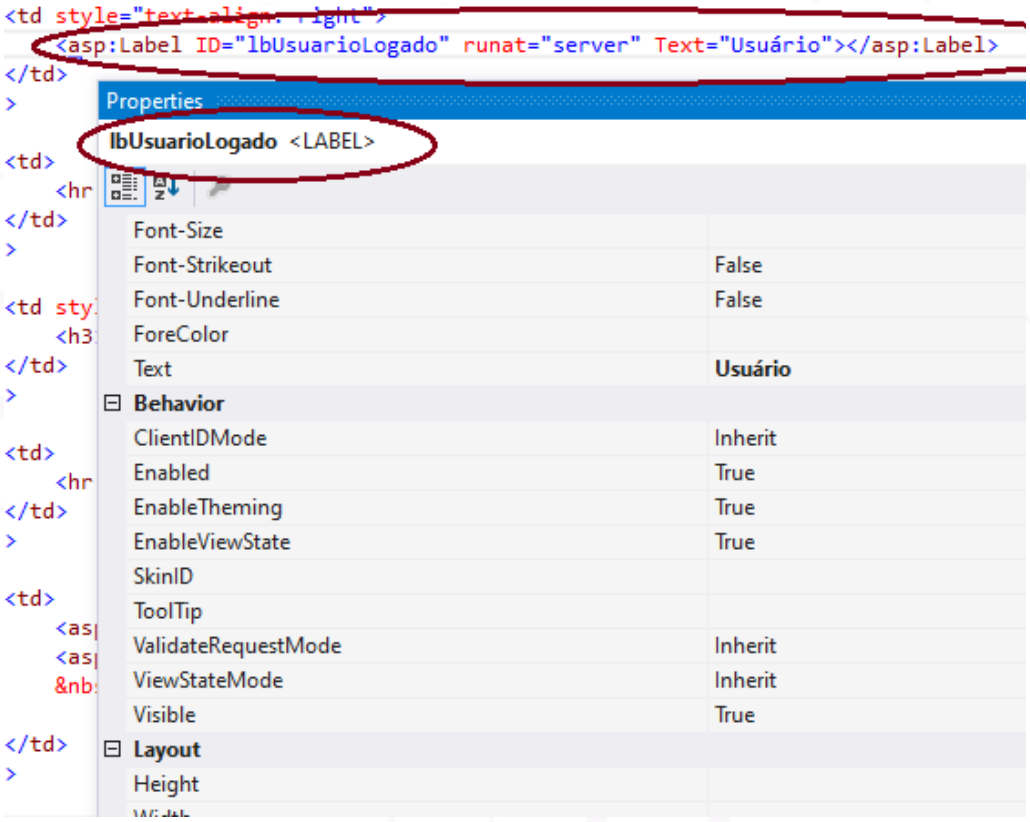
```
lbUsuarioLogado.Text = "Bem vindo " + usuario.nome;
```

5.1.6 Fechando o tag: ***</asp:Label>***

É um padrão fechar os *toquens* desta forma, mas eles podem também ser fechados de forma resumida, assim como os *toquens* HTML normais. Apesar do asp.net gerar o fechamento da forma clássica, podemos editar o fechamento para que o código fique menos poluído suprimindo o </asp:Label> apenas por uma barra antes de fechar o toquem de abertura...

```
<asp:Label ID="lbUsuarioLogado" runat="server" Text="Usuário" />
```

Existem muitas outras propriedades dos objetos gráficos que podem ser atribuídas diretamente no form.aspx ou utilizando assistentes clicando sobre o objeto com o botão direito e acessando/editando na janela de propriedades;



5.1.7 Qualquer elemento HTML pode se acessado pelo Server (Back End)

É possível transformar objetos HTML em objetos acessados via *CodeBehind* assinalando o componente com a propriedade runat="server" e um id="xxx". Assim podemos, via *Back End* acessar e alterar as propriedades do componente sem necessidade de usar javascript.

Exemplo:

```
<table style="width: 1000px;" runat="server" id="tabela01">
```

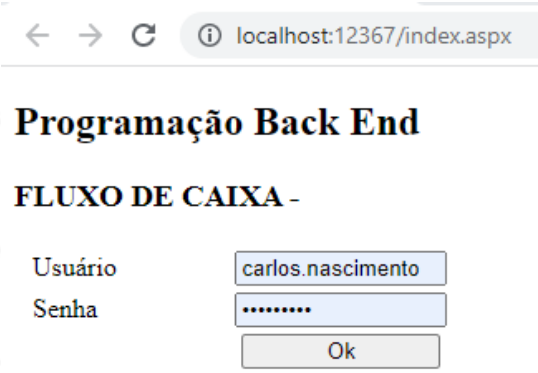
No *CodeBehind* podemos esconder esta tabela simplesmente fazendo:

```
tabela01.Visible = false;
```

5.2 A tela de login

Observando as especificações do sistema descrita no tópico 3, o Form de login deve ser projetado para o login do usuário, troca de senha de abertura (primeiro acesso) e funcionar como menu para os usuários de perfil não administrativo. O recurso utilizado foi o de tornar visível ou não os sub formulários que atendem a cada caso. Todo o código de login, troca de senha e menu foi escrito no Form index.aspx e na sua respectiva classe *CodeBehind* index.aspx.cs.

5.2.1 A tela de login básica



5.2.2 Redefinindo a senha inicial

Programação Back End

FLUXO DE CAIXA -

Usuário pedro.silva
Senha ••••••••••••
Ok

Redefinição de Senha

Nova senha ••••••••
Confirmação ••••••••
Ok

Este é o seu primeiro acesso, favor definir uma nova senha com no mínimo 6 e no máximo 10 caracteres

5.2.3 Menu de entrada (Visível apenas por usuários perfil USU)

localhost:12367/index.aspx

Programação Back End

FLUXO DE CAIXA - Universidade Santa Cecilia

Menu

Cadastro de Usuários
Fluxo de Caixa
Sair

5.2.4 A Tela de Cadastro de Usuários

localhost:12367/cadastroUsuarios.aspx

Bem vindo: Jose Alberto

Cadastro de Usuários

Buscar usuário por Nome/Login ou Sequencial Buscar

Nome* Login* CPF* Perfil*: ADM USU Incluir

Universidade Santa Cecilia

Seq.	Nome	Login	Senha	Perfil	CPF
0012	Ana Maria Do Rosário	ana.maria	21987654321	USU	21987654321
0007	Carlos Nascimento	carlos.nascimento	Já trocada	ADM	00000000000
0008	Jose Alberto	jose.alberto	Já trocada	ADM	32368280778
0009	Maria Aparecida	maria.aparecida	11111111111	ADM	11111111111
0011	Pedro Da Silva	pedro.silva	12345678912	USU	12345678912

Sair

5.2.5 A Tela Livro Caixa

Importante

Verifique/valide comparando as especificações da tela com os *Front End* apresentados. É muito importante a aderência delas ao projeto em construção. A atualização das especificações deve ser rigorosa sempre que for necessário alguma alteração no momento da codificação. O Projeto está longe de ser uma “burocracia”, ele precisa representar fielmente a realidade ou não terá nenhum valor prático.

6. Login e Menu (Back End)

Objetivo:

Explicar a construção do código.

Introdução:

Seguindo a sequência lógica do projeto, vamos primeiramente estudar o código da funcionalidade de login. Assumimos aqui que o aluno possui algum conhecimento prévio da Linguagem C# e de programação orientada a objetos. Neste ponto recomenda-se que, caso o aluno se sinta inseguro com a linguagem, procure não deixar passar nenhuma dúvida indo buscar o conhecimento necessário no material específico de C#. O Código completo do projeto está a disposição do aluno nos anexos. No tópico 4, apresentamos as classes ***Usuario***, ***Lancamento*** e ***LivroCaixa*** que serão citadas neste e nos próximos tópicos (6, 7 e 8)

6.1 O Form index.aspx

Procurando reduzir o ***Front End*** ao mínimo necessário, utilizaremos o formulário de entrada *default* do sistema, o index.aspx, para construir nossa tela de login.

6.1.1 O Panel pnLogin

```
<asp:Panel runat="server" ID="pnLogin" Visible="true">
    <table style="width: 400px">
        <tr>
            <td>  <asp:Label ID="lbUsuario" runat="server" Text="Usuário"></asp:Label>
            </td>
            <td>  <asp:TextBox ID="txUsuario" runat="server" Width="120px"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>  <asp:Label ID="lbSenha" runat="server" Text="Senha"></asp:Label>
            </td>
            <td>  <asp:TextBox ID="txSenha" runat="server" TextMode="Password" Width="120px"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td> </td>
            <td>   <asp:Button ID="btOk" runat="server" Text="Ok" OnClick="btOk_Click" Width="120px" />
            </td>
        </tr>
    </table>
</asp:Panel>
```

O **Panel** é um componente gráfico utilizado como *container* que pode, entre outras coisas, ser ocultado / mostrado em tempo de execução. A construção deste código pode ser feita diretamente ou utilizando o editor de HTML que pode ser acessado clicando no botão *Design* que aparece abaixo e a esquerda da tela.

```
<asp:Panel runat="server" ID="pnTrocaSenha" Visible="false">
    <h3>Redefinição de Senha</h3>
```

100 % Design Split Source <html> <body> <form#login> <asp:Panel#pnLogin> <table>

Como podemos observar, no trecho de código, definimos basicamente cinco objetos. Dois *labels*, duas caixas de texto (uma delas do tipo *password*) e um botão para confirmação do login. No objeto ***btOk*** da classe ***Button***, observamos a chamada do método ***btOk_Click*** que corresponde no *CodeBehind* (*Back End*) ao código da figura;

6.1.2 O método btOk_Click()

Ao clicar no botão btOk, o método *btOk_Click()* será chamado e executado proceder a verificação do usuário e seu consequente login se for o caso;

Seguindo o código do método *btOk_Click()*, observamos na primeira linha que ocultamos o ***pnTrocaSenha*** porque ele só deve aparecer nos casos de primeiro acesso do usuário com sua senha inicial. A seguir, limpamos a área de mensagens e procedemos a criação e abertura da conexão, uma vez que será necessário ir ao Banco de Dados para verificar se o usuário está cadastrado ou não. Logo abaixo da abertura de conexão, chamamos o método:

```
Usuario usuario = Usuario.busca(txUsuario.Text.Trim(), txSenha.Text.Trim(), con);
```

Os argumentos passados são as propriedades *text* dos objetos gráficos definidos no Form index.aspx: *txUsuario* e *txSenha*. Na classe *CodeBehind* do Form, eles são referenciados como objetos em cuja propriedade text estão armazenados o que o usuário digitou na interface para login e senha. Por uma questão de segurança, para evitar práticas como SQL Injection, sempre que precisamos utilizar campos digitados pelos usuários devemos trabalhar com preparação de queries. Desta forma ficamos tranquilos que as tentativas de invasão via *sql ingection* não terão sucesso. Veja (no método busca) que login e senha são passados como argumentos para o método *Conexao.executaSelect().*

Saiba mais

Pesquise o que é e como se proteger de SQL Injection, a segurança de aplicações web é particularmente crítica uma vez que, qualquer pessoa conectada na web, pode acessar nosso endereço. A tela mais crítica é a de login uma vez que o usuário ainda não foi registrado. Mesmo assim, recomenda-se fortemente que

todas as queries com campos digitados pelo usuário utilizem o recurso de preparação de queries.

```
protected void btOk_Click(object sender, EventArgs e)
{
    pnTrocaSenha.Visible = false;
    lbMensagem.Text = String.Empty;
    using (Conexao con = new Conexao(null))
    {
        con.open();
        Usuario usuario = Usuario.busca(txUsuario.Text.Trim(), txSenha.Text.Trim(), con);
        if (usuario == null)
        {
            lbMensagem.Text = "Usuário / senha não cadastrado!";
            return;
        }
        Session["usuario"] = usuario;
        if (usuario.senha == usuario.cpf)  // Primeiro acesso
        {
            lbMensagem.Text = "Este é o seu primeiro acesso, favor definir" +
                " uma nova senha com no mínimo 6 e no máximo 10 caracteres";
            pnTrocaSenha.Visible = true;
            return;
        }
        if (usuario != null && usuario.senha != usuario.cpf)
        {
            HttpCookie cookie = Request.Cookies["empresa"];
            if (cookie == null || cookie.Value.ToString() == String.Empty)
            {
                boxOkCancelaTextoJavaScript("Digite o nome da instituição", "digiteNome");
            }
            else
            {
                Session["empresa"] = empresa = cookie.Value;
                pnLogin.Visible = false;
                if (usuario.perfil == "ADM")
                {
                    pnMenu.Visible = true;
                }
                else
                {
                    Response.Redirect("fluxoDeCaixa.aspx", false);
                    return;
                }
            }
        }
    }
}
```

6.1.3 Dando uma expiada no código do método busca:

```
public static Usuario busca(String login, String senha, Conexao con)
{
    try
    {
        string []args = {login, senha};
        String sql = String.Concat("SELECT ", campos, ",idUsuario FROM ", nomeTabela,
            " WHERE login=@1  AND senha=@2");
        DataTable dt = Conexao.executaSelect(con, sql,args);
        if (dt.Rows.Count == 0) return null;
        DataRow[] r = dt.Select();
        Usuario item = new Usuario(r[0][0].ToString(), r[0][1].ToString(), r[0][2].ToString(),
            r[0][3].ToString(), r[0][4].ToString());
        item.idUsuario = Int16.Parse(r[0][5].ToString());
        return item;
    }
    catch (Exception)
    {
        throw;
    }
}
```

A *query* sql, basicamente procura um usuário no Banco que corresponda ao conjunto login e senha recebidos como argumento. Caso encontre, retorna o objeto da classe *Usuario* populado, caso contrário retorna uma referência null, indicando que não encontrou. Se encontrando, verifica se a senha digitada corresponde ao CPF do usuário. Caso positivo, significa que se trata do primeiro acesso. Mostra o painel ***pnTrocaSenha*** e aguarda que o usuário cadastre a nova senha. O usuário digita a nova senha duas vezes, o sistema compara para verificar se são idênticas e que estão na conformidade de dígitos, entre 6 e 10. Tudo estando OK, atualiza os dados na tabela de usuários e libera o acesso ao sistema;

6.1.4 Trabalhando com Cookies

```
HttpCookie cookie = Request.Cookies["empresa"];
if (cookie == null || cookie.Value.ToString() == String.Empty)
{
    boxOkCancelaTextoJavaScript("Digite o nome da instituição", "digiteNome");
}
else
{
    Session["empresa"] = empresa = cookie.Value;
    pnLogin.Visible = false;
    if (usuario.perfil == "ADM")
    {
        pnMenu.Visible = true;
    }
    else
    {
        Response.Redirect("fluxoDeCaixa.aspx", false);
        return;
    }
}
```

Uma vez registrado o usuário, o sistema verifica se existe um *cookie* cadastrado chamado *"empresa"*. Caso o *cookie* ainda não esteja indefinido, abre uma caixa de diálogo do navegador solicitando a definição do nome da instituição que opera o sistema, caso contrário coloca na sessão o nome da instituição para ser usado nas demais telas;

Neste ponto o sistema verifica se o usuário é perfil ADM, se for, mostra o painel do menu, caso contrário redireciona para o Form FluxoDeCaixa.aspx

```
pnLogin.Visible = false;
if (usuario.perfil == "ADM")
{
    pnMenu.Visible = true;
}
else
{
    Response.Redirect("fluxoDeCaixa.aspx", false);
    return;
}
```

No caso de usuário perfil ADM, o painel do menu fica *visible*:

```
<asp:Panel runat="server" ID="pnMenu" Visible="false">
    <table style="width: 300px">
        <tr>
            <td style="text-align: center">
                <h2>Menu</h2>
            </td>
        </tr>
        <tr>
            <td style="text-align: center">
                <a href='cadastroUsuarios.aspx' title="clique aqui para cadastro de usuários.">
                    Cadastro de Usuários</a>
            </td>
        </tr>
        <tr>
            <td style="text-align: center">
                <a href='fluxoDeCaixa.aspx' title="clique aqui lançamentos do Fluxo de Caixa.">
                    Fluxo de Caixa</a>
            </td>
        </tr>
        <tr>
            <td style="text-align: center">
                <a href='index.aspx?sair=sair' title="Sair">Sair</a>
            </td>
        </tr>
    </table>
</asp:Panel>
```

E o ***Front End*** vai ter a aparência como mostra o item *5.2.3*:

Como o menu foi implementado como links, este é o código do ***Back End*** que “atende” a opção do menu Sair:

```
String sair = Request.QueryString["sair"];

if (sair != null && sair == "sair")
{
    Session["usuario"] = null;
    pnTrocaSenha.Visible = false;
    pnLogin.Visible = true;
    txUsuario.Text = txSenha.Text = String.Empty;
    Response.Redirect("index.aspx", false);
}
```

Os forms de Cadastro de Usuários e Fluxo de caixa são chamados diretamente pelos hiperlinks

Importante

Cadastre diretamente na tabela de usuários um usuário ADM para que seja possível partir o sistema já com um usuário registrado.

Use o Server Explorer para acessar e editar o usuário de partida;

7. O Cadastro de usuários (Back End)

Objetivo:

Mostrar como construir um ***CRUD*** (*Create*, *read*, *update* e *delete*) básico perfeitamente justificado no projeto**.**

Introdução:

O Cadastro de Usuários, conforme os requisitos listados no Item: *3.1.1.3,* deve permitir cadastro de novos usuários, edição de dados de usuários existentes e exclusão. No caso de novos usuários, basta digitar os dados do novo usuário e confirmar clicando no botão *Incluir*. Para editar ou excluir, é necessário fazer a busca do registro. Quando se procede uma busca, os botões de *Atualizar* e *Excluir* são mostrados. Quando o usuário cadastra um novo registro, edita ou exclui um existente, o sistema volta sempre para o modo inicial de inclusão.

7.1. Cadastrando novos usuários

A situação *default* da tela é de inserção de registros, o usuário digita os dados obrigatórios (são todos) e clica no botão de inserir. O sistema valida os campos rodando ***validaCampos().*** O método validaCampos, no caso de encontrar algum campo inconsistente, mostra a mensagem na tela e retorna um ***false*** para o evento *btNovo_Click()* que retorna sem proceder nenhuma alteração. Estando tudo ok, instancia um objeto **Usuario**, popula com os dados digitados na tela e cadastra no Banco de Dados e na sequência mostra no *grid* de usuários cadastrados;

Observe que o sistema não permite a inserção de mais de um usuário com o mesmo *login* e antes de salvar o CPF, retira os pontos e a barra por ventura digitados pelo usuário.

Saiba mais

O projeto que estamos desenvolvendo não pode ser muito complexo porque precisamos compreendê-lo em sua totalidade no intervalo de tempo que dispomos para este tópico no seu curso. Alguns detalhes estão sem cobertura, como por exemplo a validação do CPF. Pesquise métodos de validação de CPF, procurando compreender o seu funcionamento.

7.1.1 O método de inserção de novos registros: btNovo_Click()

```
protected void btNovo_Click(object sender, EventArgs e)
{
    if (!validarCampos()) return;
    Usuario usuario = (Usuario)Session["usuario"];
    try
    {
        using (Conexao con = new Conexao(usuario))
        {
            con.open();
            Usuario u = Usuario.busca(con, txLogin.Text);
            if (u != null && u.idUsuario > 0)
            {
                lbMensagem.Text = "Já existe um usuário com este login: " + txLogin.Text;
                return;
            }
            txCPF.Text = txCPF.Text.Replace(".", "").Replace("/", "");
            u = new Usuario(txNome.Text, txLogin.Text, txCPF.Text, rbAdm.Checked ? "ADM" : "USU", txCPF.Text);
            u.insere(con);
            limpa();
            lbMensagem.Text = "Novo usuário cadastrado com sucesso!";
            relatorioUsuarios.Text = usuario.montaTabela((String)Session["empresa"], con);
        }
    }
    catch (Exception e1)
    {
        if (usuario.perfil == "ADM")
        {
            lbMensagem.Text = e1.Message;
        }
        else
        {
            lbMensagem.Text = "Erro cadastrando usuario.";
        }
    }
}
```

7.1.2 O método validarCampos()

Checa cada um dos campos obrigatórios, mostrando mensagem e retornando ***false*** sempre que encontrar alguma incosistência. Tudo correndo bem, retorna ***true***;

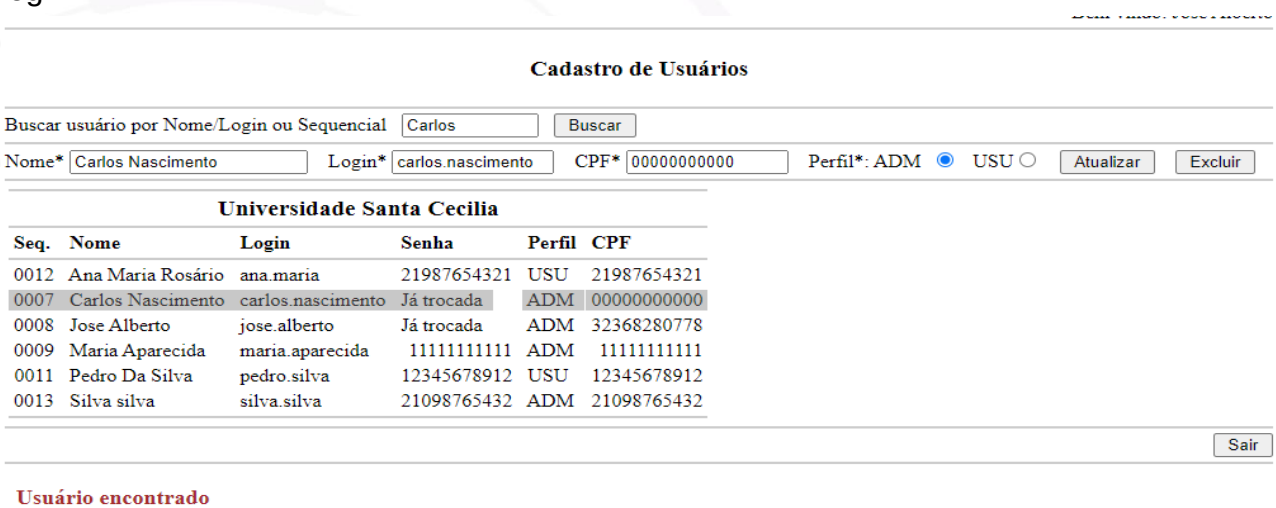
```
private bool validarCampos()
{
    if (rbAdm.Checked == false && rbUsu.Checked == false)
    {
        lbMensagem.Text = "Selecione o perfil do usuário";
        return false;
    }
    if (txNome.Text.Trim()  == String.Empty)
    {
        lbMensagem.Text = "Digite o nome do Usuário";
        return false;
    }
    if (txLogin.Text.Trim() == String.Empty)
    {
        lbMensagem.Text = "Digite o login do usuário";
        return false;
    }
    if (txCPF.Text.Trim() == String.Empty)
    {
        lbMensagem.Text = "Digite o CPF do usuário";
        return false;
    }
    return true;
}
```

7.2 Buscando

```
protected void btOkBusca_Click(object sender, EventArgs e)
{
    Usuario usuario = (Usuario)Session["usuario"];
    Usuario sel = null;
    lbMensagem.Text = String.Empty;
    btExcluir.Visible = btAtualizar.Visible = false;
    btNovo.Visible = true;
    limpa();
    Int16 id;
    if (txNomeBusca.Text.Trim() == String.Empty)  {
        lbMensagem.Text = "Digite um texto para busca (nome, login ou sequencial)";   return;
    }
    using (Conexao con = new Conexao(usuario))
    {
        con.open();
        if (Int16.TryParse(txNomeBusca.Text, out id)) // Será que o usuário digitou um número?
        {
            sel = Usuario.busca(id, con);
            if (sel == null)  {
                lbMensagem.Text = "Nenhum usuário foi encontrado com o sequencial " + id + "!";  return;
            }
        }
        else // Não é numero, então é substring
        {
            List<Usuario> lista = Usuario.listaUsuarios(txNomeBusca.Text, con); // Metodo que traz uma lista de usuários
            if (lista.Count == 0) // Lista vazia? encontrei ninguém
            {
                lbMensagem.Text = "Nenhum usuário foi encontrado com o nome/login informado!";  return;
            }
            if (lista.Count > 1) // Mais que um na lista? não vale.. preciso achar só um
            {
                lbMensagem.Text = "Mais de um usuário foi encontrado com o nome/login informado. Procure refinar melhor sua busca!";
                return;
            }
            sel = lista[0];
        }
        Session["usuarioProcurado"] = sel; // Achei
        txNome.Text = sel.nome; // Coloco os campos na tela
        txLogin.Text = sel.login;
        txCPF.Text = sel.cpf;
        if (sel.perfil == "ADM") // se for ADM, arrumo os radiobutton
        {
            rbAdm.Checked = true;
            rbUsu.Checked = false;
        }  else // se não é USU, arrumo os radiobutton
        {
            rbAdm.Checked = false;
            rbUsu.Checked = true;
        }
        lbMensagem.Text = "Usuário encontrado";
        btExcluir.Visible = btAtualizar.Visible = true;
        btNovo.Visible = false;
    }
}
```

7.2.1 Buscando um elemento já cadastrado

O sistema permite três tipos de busca, por nome, login ou pelo sequencial do usuário. No exemplo procuramos por *Carlos* que existe como "*substring*" tanto no nome quanto no login.



7.3 Editando um registro

Uma vez encontrado o registro, seu conteúdo é carregado nos respectivos campos da tela, permitindo a partir deste momento, tanto alterações quanto sua exclusão. Note que o campo senha, quando já alterado, não é mostrado no *grid* nem permite edição. O mesmo método *validarCampos()* utilizado no cadastro ne novos registros é chamado aqui também para evitar alterações inconsistentes.

```
protected void btAtualizar_Click(object sender, EventArgs e)
{
    if (!validarCampos()) return;
    Usuario usu = (Usuario)Session["usuarioProcurado"];
    if (usu == null)
    {
        lbMensagem.Text = "Erro inesperado. Usuário para editar ainda não selecionado!";
        return;
    }
    txCPF.Text = txCPF.Text.Replace(".", "").Replace("/", "");
    Usuario usuario = (Usuario)Session["usuario"];
    try
    {
        using (Conexao con = new Conexao(usuario))
        {
            con.open();
            usu.nome = txNome.Text.Trim();
            usu.login = txLogin.Text.Trim();
            usu.cpf = txCPF.Text.Trim();
            usu.perfil = rbAdm.Checked ? "ADM" : "USU";
            usu.atualiza(con);
            limpa();
            relatorioUsuarios.Text = usuario.montaTabela((String)Session["empresa"], con);
            lbMensagem.Text = "Usuário atualizado com sucesso!";
        }
    }
    catch (Exception e1)
    {
        if (usuario.perfil == "ADM")
        {
            lbMensagem.Text = e1.Message;
        }
        else
        {
            lbMensagem.Text = "Erro excluindo usuario";
        }
    }
}
```

7.4 Excluindo um registro

O evento do botão *btExcluir* utiliza o objeto encontrado na busca, que está armazenado na sessão *["usuarioProcurado"]*, para chamar o método *exclui ()* da classe ***Usuario***. Note que, na classe *CodeBehind* do Form *CadastroUsuarios.aspx***,** não se faz acesso direto ao banco de dados, ficando todo este serviço a cargo das classes entidade, que são as classes projetadas para espelhar cada uma das tabelas entidade no Banco de Dados (***Usuario*** e ***Lancamento***).

```
protected void btExcluir_Click(object sender, EventArgs e)
{
     Usuario usuario = (Usuario)Session["usuario"];
    try
    {
        Usuario usu = (Usuario)Session["usuarioProcurado"];
        if (usu == null)
        {
            lbMensagem.Text = "Erro inesperado. Usuário para excluir ainda não selecionado!";
            return;
        }
        using (Conexao con = new Conexao(usuario))
        {
            con.open();
            usu.exclui(con);
            relatorioUsuarios.Text = usuario.montaTabela((String)Session["empresa"], con);
        }
        limpa();
        lbMensagem.Text = "Usuário excluido com sucesso!";
    }
    catch (Exception e1)
    {
        if (usuario.perfil == "ADM")
        {
            lbMensagem.Text = e1.Message;
        }
        else
        {
            lbMensagem.Text = "Erro excluindo usuário";
        }
    }
}
```

8. O Livro Caixa (Back End)

Objetivo:

Compreender o projeto explorando as regras de negócio características de um livro caixa.

Introdução:

Esta é a principal funcionalidade do sistema e foi projetada para controlar o fluxo de caixa. Ele está preparado para operar por períodos mensais que podem ser fechados fora do último dia do mês se necessário. Quando o usuário fechar o período, o sistema cria um novo, transportando o saldo do período que foi fechado para o novo que está abrindo. Os períodos anteriores podem ser consultados livremente mas não podem receber novos lançamentos. Neste item vamos estudar o código desenvolvido.

8.1 Usando Javascript

Temos no projeto uma oportunidade para mostrar o uso de uma função *Javascript* chamada no campo texto onde digitamos o valor do lançamento. Esta função é muito útil porquê além de formatar o campo valor com duas casas decimais, facilita a consistência impedindo que o usuário digite caracteres não numéricos. Normalmente o código Javascript é carregado na página diretamente de um ou mais arquivos extensão *.js* que devem ser anexados ao projeto. No nosso caso, o código *Javascript* foi incorporado diretamente no Form *FluxoDeCaixa.aspx* para facilitar o estudo. O estudo de *Javascript* está mais voltado ao estudo e desenvolvimento do *Front End,* mas como já dissemos, não é possível falar de *Back End* sem se referir ao *Front End*.

8.1.1 Chamando o método Javascript

Veja que na caixa de texto *txValor*, estamos associando o evento *onkeyup* a nossa função Javascript *formataValor ()*

```
<asp:TextBox ID="txValor" runat="server"  Width="100px" onkeyup='formataValor(this,event);' />
```

Os argumentos passados são a própria caixa de texto (this) e o evento.

8.1.2 O código da função formataValor()

```
function formataValor(campo, evt) {
    var xPos = PosicaoCursor(campo);
    evt = getEvent(evt);
    var tecla = getKeyCode(evt);
    if (!teclaValida(tecla))  return;
    vr = campo.value = filtraNumeros(filtraCampo(campo));
    if (vr.length > 0) {
        vr = parseFloat(vr.toString()).toString();
        tam = vr.length;

        if (tam == 1) campo.value = "0,0" + vr; if (tam == 2) campo.value = "0," + vr;
        if ((tam > 2) && (tam <= 5)) campo.value = vr.substr(0, tam - 2) + ',' +
            vr.substr(tam - 2, tam);
        if ((tam >= 6) && (tam <= 8)) campo.value = vr.substr(0, tam - 5) +
            '.' + vr.substr(tam - 5, 3) + ',' + vr.substr(tam - 2, tam);
        if ((tam >= 9) && (tam <= 11)) campo.value = vr.substr(0, tam - 8) +
            '.' + vr.substr(tam - 8, 3) + '.' + vr.substr(tam - 5, 3) + ','
            + vr.substr(tam - 2, tam);
        if ((tam >= 12) && (tam <= 14)) campo.value = vr.substr(0, tam - 11)
            + '.' + vr.substr(tam - 11, 3) + '.' + vr.substr(tam - 8, 3) + '.' +
            vr.substr(tam - 5, 3) + ',' + vr.substr(tam - 2, tam);
        if ((tam >= 15) && (tam <= 18)) campo.value = vr.substr(0, tam - 14)
            + '.' + vr.substr(tam - 14, 3) + '.' + vr.substr(tam - 11, 3) + '.' +
            vr.substr(tam - 8, 3) + '.' + vr.substr(tam - 5, 3) + ',' +
            vr.substr(tam - 2, tam);
    }
    MovimentaCursor(campo, xPos);
}
```

Todas as chamadas de funções destacadas em amarelo, são códigos de funções *Javascript* que foram escritas na página *FluxoDeCaixa.aspx* e que o aluno pode examinar consultando o código do projeto.

8.2 A tela de lançamentos

Bem vindo Jose Alberto

Fluxo de Caixa

Selecione o período: Ano/Mes 2022/08 Situação: **Aberto**

Universidade Santa Cecilia

Seq.	Descrição	Responsável	Crédito	Débito	Data	Saldo
	Valor Transportado do mês anterior		2.492,74			2.492,74
0021	Compra de Grampeador para Dna. Lúcia	Jose Alberto		145,76	01/08/2022 10:20	2.346,98
0022	Lanche do Sr. Carlos	Jose Alberto		23,49	01/08/2022 10:21	2.323,49
0023	Pagamento dos serviços de eletricista. Troca de lâmpadas sala de reunião.	Jose Alberto		200,00	01/08/2022 10:23	2.123,49
0024	Tonner para impressora	Jose Alberto		565,99	04/08/2022 04:18	1.557,50
0025	Receita de venda de balcão	Jose Alberto	234,88		04/08/2022 04:19	1.792,38

Descrição* Débito* ○ Crédito* ○ Valor* Lançar

Fechar Período Sair

8.2.1 Inserindo novos lançamentos

Os campos descrição, um dos Radio Buttons se Débito ou Crédito e o valor do lançamento, são campos obrigatórios. O usuário logado e a data atual são fornecidos diretamente pelo sistema.

8.2.1.1 O evento btLancar_Click()

Como os lançamentos não podem ser editados nem excluídos, é importante que o usuário tenha chance de verificar os dados do lançamento antes de prosseguir. Por isso mostramos uma caixa de confirmação. O método valida(), logo no início do evento, aborta o processo de inserção de lançamentos caso exista alguma inconsistência

```
protected void btLancar_Click(object sender, EventArgs e)
{
    if (!valida()) return;

    StringBuilder myScript = new StringBuilder(String.Empty);
    myScript.Append("<script type='text/javascript' language='javascript'> ");
    myScript.Append("var result = window.confirm('Confirma realizar o lançamento? Valor: R$" +
        txValor.Text + " - " + (rbCredito.Checked ? "Crédito":"Débito") + "'); ");
    myScript.Append("__doPostBack('lancamento', result); ");
    myScript.Append("</script> ");
    ScriptManager.RegisterStartupScript(Page, Page.GetType(), "msg", myScript.ToString(), false);
}
```

8.2.1.2 O método valida();

```
private bool valida()
{
    if (txDescricao.Text.Trim() == String.Empty)
    {
        lbMensagem.Text = "Descrição é campo obrigatório!";
        return false;
    }

    if (txValor.Text.Trim() == String.Empty)
    {
        lbMensagem.Text = "Valor é campo obrigatório!";
        return false;
    }

    double valor;

    if (!Double.TryParse(txValor.Text, out valor))
    {
        lbMensagem.Text = "Valor digitado não é um número válido!";
        return false;
    }

    if (valor <= 0)
    {
        lbMensagem.Text = "Valor digitado deve ser maior que zero e positivo!";
        return false;
    }

    if (rbCredito.Checked == false && rbDebito.Checked == false)
    {
        lbMensagem.Text = "Selecione se o lançamento é de crédito ou débito";
        return false;
    }

    if (rbDebito.Checked && valor > livro.saldo)
    {
        lbMensagem.Text = "Saldo em caixa insuficiente para retirada!";
        return false;
    }

    return true;
}
```

Entendendo

Uma das principais preocupações do desenvolvedor *Back End* é a validação do que o usuário digita. Esta validação pode ser feita no código do *Front End* mas, mesmo assim, devem ser replicadas no código *Back End* mesmo que pareçam desnecessárias. A função *Javascript* chamada no evento do campo *txValor*, pode garantir que o valor digitado será numérico. Mesmo assim, o *Back End* valida o campo. Lembre-se que o *Javascript* roda no cliente e pode ser "*bypassado*".

8.2.1.3 Aparência da tela depois de concluído o lançamento:

Bem vindo Jose Alberto

Fluxo de Caixa

Selecione o período: Ano/Mes 2022/08 Situação: **Aberto**

Universidade Santa Cecilia

Seq.	Descrição	Responsável	Crédito	Débito	Data	Saldo
	Valor Transportado do mês anterior		2.492,74			2.492,74
0021	Compra de Grampeador para Dna. Lúcia	Jose Alberto		145,76	01/08/2022 10:20	2.346,98
0022	Lanche do Sr. Carlos	Jose Alberto		23,49	01/08/2022 10:21	2.323,49
0023	Pagamento dos serviços de eletricista. Troca de lâmpadas sala de reunião.	Jose Alberto		200,00	01/08/2022 10:23	2.123,49
0024	Tonner para impressora	Jose Alberto		565,99	04/08/2022 04:18	1.557,50
0025	Receita de venda de balcão	Jose Alberto	234,88		04/08/2022 04:19	1.792,38
0026	Ponte aérea para Dr. Gilberto	Jose Alberto		867,08	05/08/2022 04:58	925,30

Descrição* Débito* Crédito* Valor* Lançar

Fechar Período Sair

```
String evento = (this.Request["__EVENTTARGET"] == null) ? String.Empty : this.Request["__EVENTTARGET"];
if (evento == "lancamento")
{
    Lancamento l = new Lancamento(txDescricao.Text, livro.idLivroCaixa, Double.Parse(txValor.Text),
                                  rbCredito.Checked ? 'C' : 'D', usuario, DateTime.Now);
    l.insere(con);
    livro.listaLancamentos(con);
    txValor.Text = txDescricao.Text = String.Empty;
    rbDebito.Checked = rbCredito.Checked = false;
}
```

A resposta do usuário dada a Caixa de Confirmação, precisa ser capturada no *post back* do formulário. Isso é feito como mostra o fragmento de código inserido no método *Page_Load* do Form *FluxoDeCaixa.aspx*.

No *PostBack* do formulário, caso o usuário tenha clicado no *OK*, o novo lançamento é instanciado e populado usando o construtor da classe e, em seguida, inserido no Banco de Dados. O *grid* já atualizado, contendo os lançamentos do período é mostrado e os campos onde foram digitados os dados são inicializados com string vazio. Caso o usuário tenha clicado no *Cancelar*, o string evento obtido do Request[“__EVENTTARGET”], retorna vazio e o processo de inserção de novo lançamento é abortado.

*8.2.1.4 O código do método **insere()** da classe **Lancamento**;*

```
public int insere(Conexao con)
{
    try
    {
        String []args = {descricao};
        String sql = String.Concat("INSERT INTO ", nomeTabela, " (", campos, ") Values (@1,'", idLivroCaixa, "','",
            (valor+"").Replace(",",".."), "','", tipoLancamento , "','", usuarioResponsavel.idUsuario, "','",
            data.ToString("dd/MM/yyyy hh:mm:ss") , "')");
        idLancamento = Conexao.executaQuery(con, sql, nomeTabela,args);
        return idLancamento;
    }
    catch (Exception)
    {
        throw;
    }
}
```

Observe o cuidado com o *Sql Injection* aplicado apenas a descrição do lançamento, já que é o único campo texto editável pelo usuário.

8.3 Fechando o período.

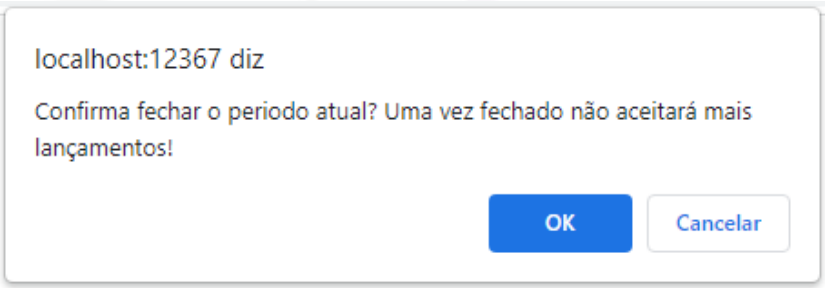
Nos requisitos do projeto, item *3.1.1.4*, subitens *f* e *g*, o projeto especifica que os períodos de lançamentos são definidos pelo mês/ano selecionado e esta ação seria feita pelo usuário em um dia arbitrário do período. Isto foi implantado inserindo novos registros na tabela LivroCaixa.

8.3.1 Fechando o período atual – analisando o código

Ao clicar no botão de *btFecharPeriodo*, o evento *btFecharPeriodo_Click* é executado:

```
protected void btFecharPeriodo_Click(object sender, EventArgs e)
{
    StringBuilder myScript = new StringBuilder(String.Empty);
    myScript.Append("<script type='text/javascript' language='javascript'> ");
    myScript.Append("var result = window.confirm('Confirma fechar o periodo atual? Uma vez " +
        "fechado não aceitará mais lançamentos!'); ");
    myScript.Append("__doPostBack('fecharPeriodo', result); ");
    myScript.Append("</script> ");
    ScriptManager.RegisterStartupScript(Page, Page.GetType(), "msg", myScript.ToString(), false);
}
```

É mostrada para o usuário a caixa de mensagem do navegador:



De forma análoga ao que vimos no *Cadastro de Usuários*, precisamos tratar o retorno do *post back* da resposta a caixa de diálogo.

```
String evento = (this.Request["__EVENTTARGET"] == null) ? String.Empty : this.Request["__EVENTTARGET"];
if (evento == "fecharPeriodo")
{
    livro.fechaPeriodo(con);
    btFecharPeriodo.Enabled = false;
    comboAnoMes.Items.Clear();
    montaComboAnoMes(comboAnoMes, con);
}
```

Atendendo o evento para o caso do usuário clicar em *OK*, o método *fecharPeriodo* da classe ***LivroCaixa*** é chamado. Em seguida o botão de fechar caixa é desabilitado, o combo de períodos é limpo e logo em seguida remontado para que aconteça a sua atualização.

*8.3.2 O método **fecharPeriodo()** da classe **LivroCaixa***

Vamos analisar o código de como é efetivamente feito o fechamento do período. A primeira providência é iniciar a transação, fazemos isso porque estamos alterando/criando dois registros em momentos diferentes. Para garantir que não teremos inconsistências no par de registros, usamos a transação para garantir a integridade dos dados. Na primeira *query*, alteramos o *flag stLivroFechado* para 1 sinalizando que o registro atual está sendo fechado. Em seguida, montamos uma data, *proximoPeriodo*, que aponta sempre para o mês seguinte ao mês do período que está sendo fechado. A segunda *query*, insere um novo registro no Banco de Dados. Finalmente a transação é finalizada e os dados são fisicamente atualizados no banco com o *commit()*. No caso de erro, o tratamento de exceção deixa os registros no banco inalterados com o *rollback ()*.

```
1 reference
public void fechaPeriodo(Conexao con)
{
    try
    {
        con.beginTransaction(); // Afeta dois registros, por isso foi implementada transação
        string sql = String.Concat("UPDATE " , nomeTabela , " SET stLivroFechado=1 Where " , nomeId , "=" ,
            idLivroCaixa);
        Conexao.executaQuery(con, sql, nomeTabela, null);
        DateTime proximoPeriodo = new DateTime(Int16.Parse(ano), Int16.Parse(mes), 1).AddMonths(1);
        sql = String.Concat("INSERT INTO  ", nomeTabela, " (", campos, ") Values (", (saldo + "").Replace(",", ."),",",
            proximoPeriodo.Month , ",", proximoPeriodo.Year , ",0)");
        Conexao.executaQuery(con, sql, nomeTabela,null);
        con.commit();
    }
    catch (Exception)
    {
        con.rollback();
        throw;
    }
}
```

9. Colocando a aplicação no ar

Objetivo:

Mostrar como colocar no ar uma aplicação *WEB*, discutindo detalhes e servidor de aplicação e de banco de dados

Introdução:

Durante o desenvolvimento e testes de um sistema WEB, é bastante prático utilizar o servidor local IIS (Serviços de Informações da Internet). Com ele, temos o comportamento idêntico do sistema que estaria em um site da internet. Para hospedar nossa aplicação na WEB, precisamos gerar um executável e colocar abaixo de um servidor, com um endereço físico para que possa ser acessado de qualquer lugar do planeta.

9.1 O Servidor IIS

Segundo a definição da Microsoft, o IIS é um servidor web flexível e seguro para hospedar qualquer coisa na WEB. De *streaming* de mídia a aplicações WEB, a arquitetura do IIS é escalável e aberta projetada para ser robusta e confiável. Encontramos tudo que precisamos saber sobre o IIS no endereço http://www.IIS.net. Apesar de ser um produto Microsoft, veja que este site não está hospedado no site da Microsoft. Isso porque a Microsoft trabalha para que todo tipo de aplicação WEB possa rodar abaixo dele, incluindo aplicações Java e PHP.

9.2 Os serviços de hospedagem

Existem vários serviços que hospedam aplicações e os mais conecidos são, Amazon, Heroku, Microsoft Azure e IBM Cloud.

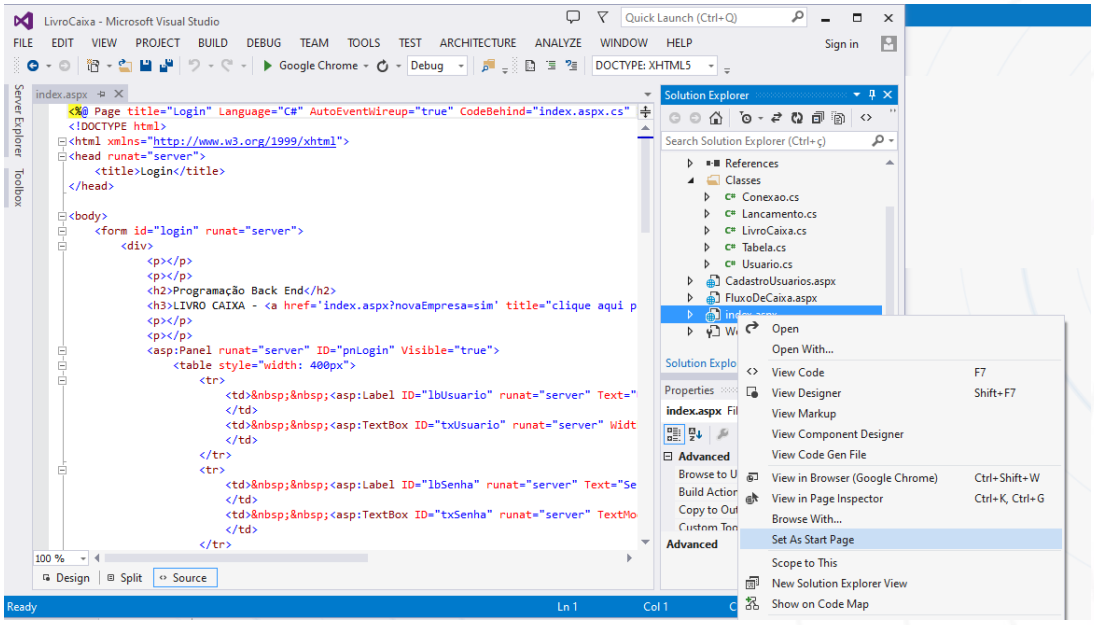
O custo do serviço sempre leva em conta desempenho, quantidade de requisições, espaço utilizado pelo banco de dados etc. Amazon, IBM e heroku por exemplo, estabelecem um limite que, se não ultrapassado, podemos hospedar um aplicativo web sem custo.

O banco de dados também será hospedado em um desses serviços e estará disponível na rede, sempre considerando um limite de acessos para efeito de cobrança. Amazon, Azure e IBM permitem que se criem subredes protegendo o banco de acessos externos.

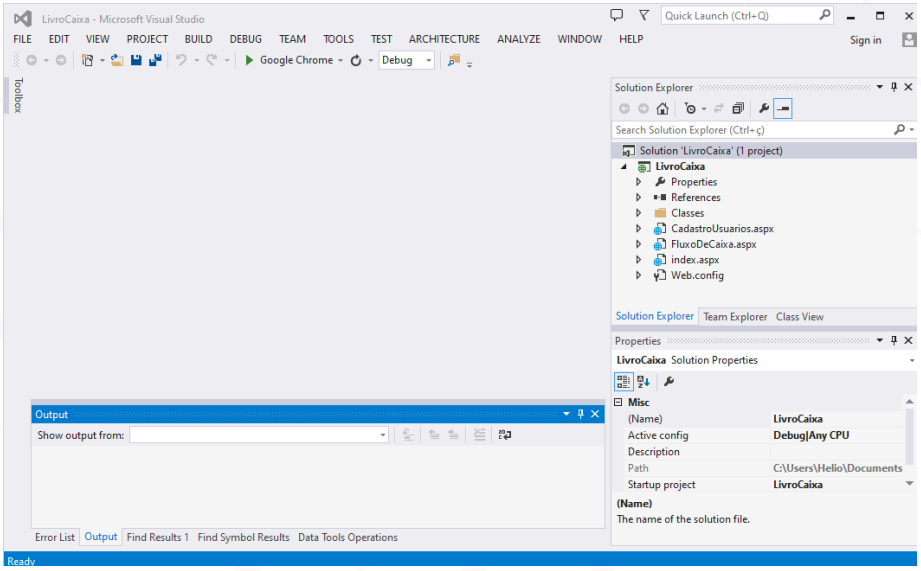
As atualizações da aplicação são feitas utilizando-se o ambiente de desenvolvimento onde compilamos a aplicação e copiamos o execuável para a pasta do servidor que irá rodar o site.

9.3 Rodando a aplicação Local Host

Abra o Visual Studio e carregue o projeto anexo ao curso selecionando o arquivo de projeto *LivroCaixa.sln*

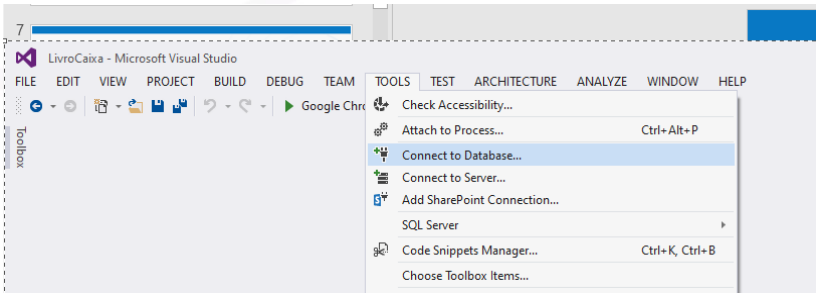


O projeto deve abrir com esta aparência:

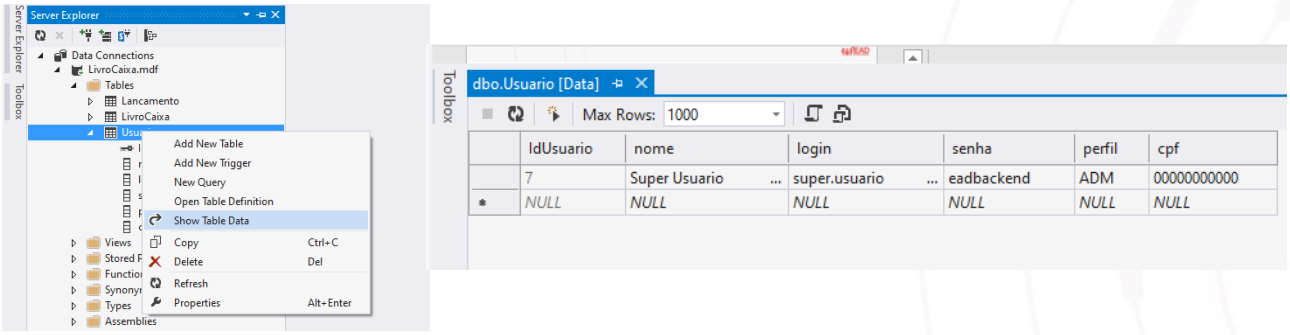


Antes de tentar rodar a aplicação, precisamos cadastrar um usuário diretamente no banco de dados para que seja possível se logar pela primeira vez.

Para acessar o banco de dados selecione *TOOLS* e depois *Conect to Database*:

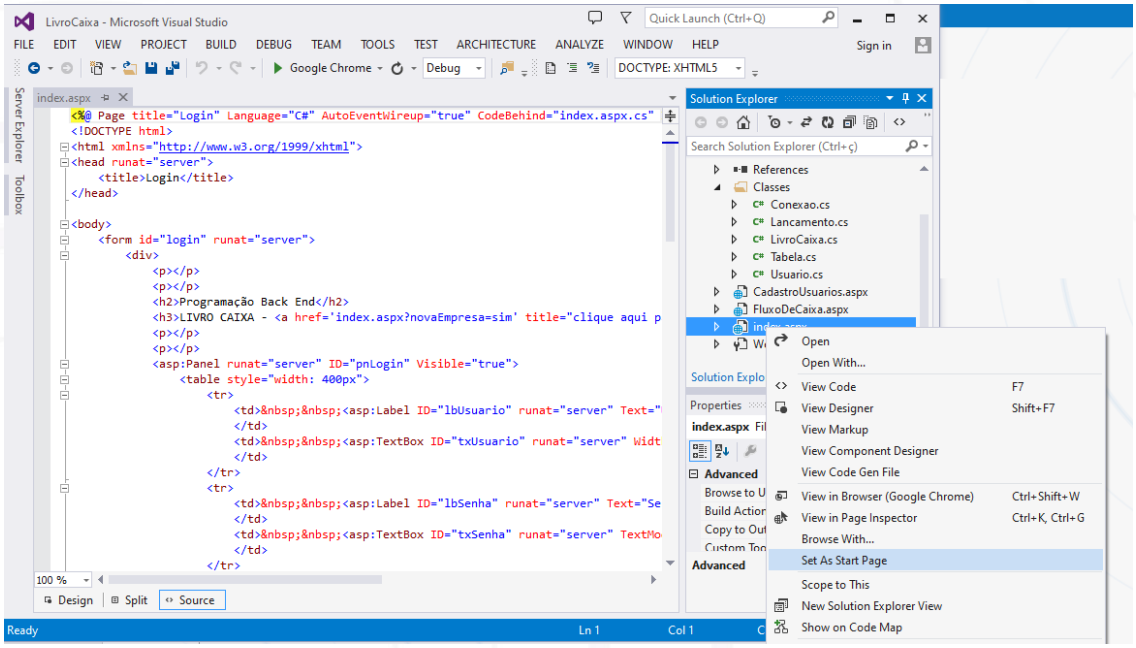


Em seguida a aba Server Explorer ficará visível permitindo acessar o Bando de Dados que já está criado mas ainda não populado.



Tudo dando certo até aqui, vamos rodar nossa aplicação pela primeira vez;

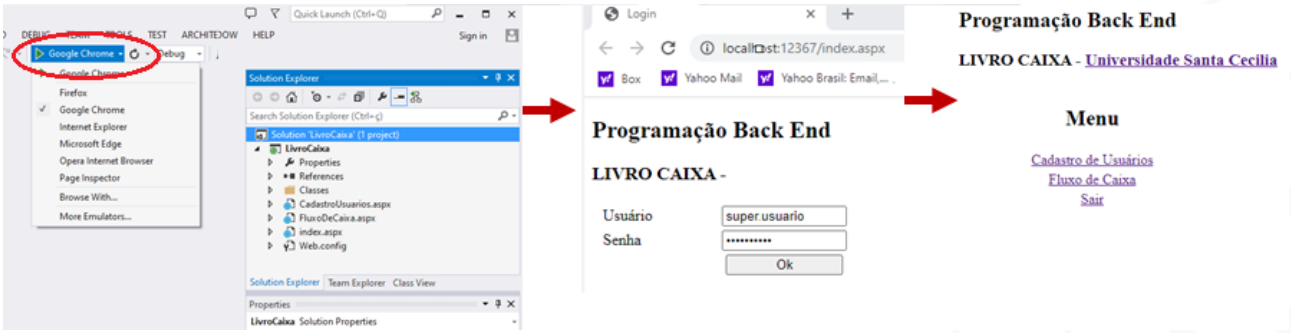
Abra o arquivo index.aspx e com botão direito do *mouse* e configure para que ele seja a página inicial da aplicação.



Para rodar a aplicação utilizando o *Local Host*, selecionamos o navegador no caso, o Google Chrome (assinalado em vermelho).

Clique sobre o nome do navegador selecionado, o asp.net irá compilar, executar e a tela de login deve aparecer

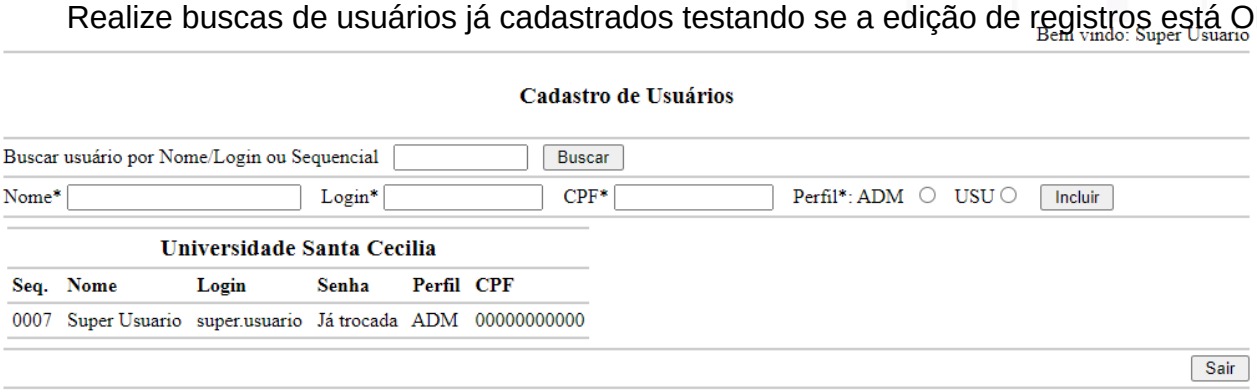
Como o usuário que cadastramos '*super.usuario*' é perfil ADM, a tela de Menu deverá ficar acessível:



9.4 Cadastrando Usuários

Cadastre alguns usuários, alternando entre os perfis de usuários ADM e USU.

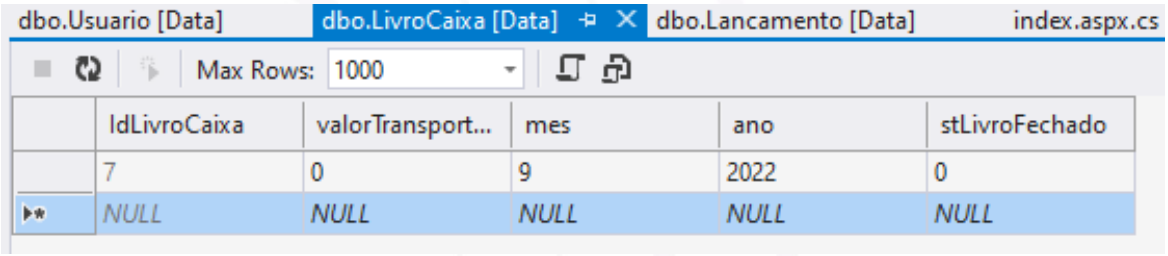
Realize buscas de usuários já cadastrados testando se a edição de registros está OK



9.5 A Tela Fluxo de Caixa

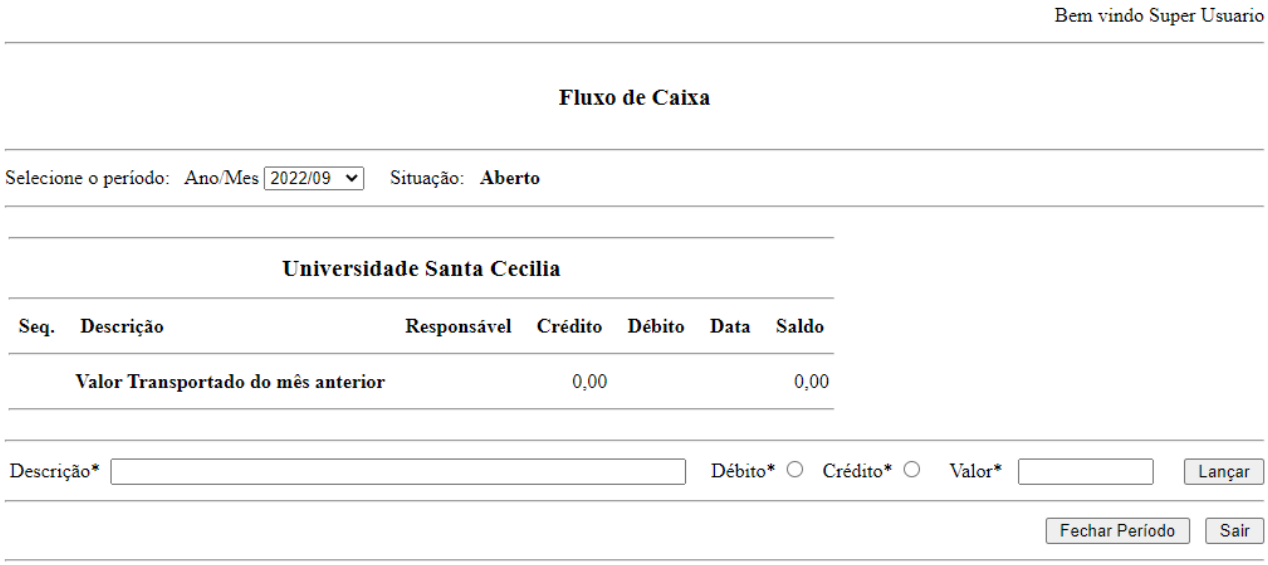
Antes de carregar a tela Fluxo De Caixa pela primeira vez, verifique se existe pelo menos um período cadastrado na tabela LivroCaixa.

Para o flag *stLivroFechado*, 0 (zero) sigifica false e 1 (um) significa true;



IdLivroCaixa	valorTransport...	mes	ano	stLivroFechado
7	0	9	2022	0
NULL	NULL	NULL	NULL	NULL

Esta é a aparência do Livro Caixa na primeira vez que é carregado!



Realize uma série de lançamentos alternando entre crédito e débito. Analise com atenção o funcionamento do Livro Caixa

*9.6 Dicas para aprender **MESMO** a programar*

- O aprendizado de programação só acontece com muitas horas de estudo, dedicação e prática;
- Não tenha receio de perguntar e pedir ajuda a seus colegas. Pessoas com mais conhecimento e experiência são preciosas ajudas que não devem ser disperdiçadas;
- Examine cada detalhe do projeto, não tenha pressa de passar para a próxima etapa sem que antes esteja seguro que compreendeu o que foi visto;
- Procure fazer pequenas alterações no projeto, fazendo melhorias, inserindo novos campos etc. O **Cadastro de Usuários** é um bom laboratório para fazer alterações.

  Alterar um projeto que esta rodando é uma das melhores maneiras de se aprender uma linguagem de programação;

- Se você conhece e/ou gosta de **Front End**, não se acanhe em dar uma melhorada no visual do projeto. Lembre-se que desenvolvedores **Full Stack** são bastante valorizados no mercado.