

Coleção UAB–UFSCar

Sistemas de Informação

Sergio Donizetti Zorzo
Rafael Tomé de Souza
Ânderson Kanegae Soares Rocha

Desenvolvimento de Aplicações Web em Java





Desenvolvimento de Aplicações Web em Java





Reitor

Targino de Araújo Filho

Vice-Reitor

Adilson J. A. de Oliveira

Pró-Reitora de Graduação

Claudia Raimundo Reyes



Secretária Geral de Educação a Distância - SEaD

Aline Maria de Medeiros Rodrigues Reali

Coordenação SEaD-UFSCar

Glauber Lúcio Alves Santiago

Marcia Rozenfeld G. de Oliveira

Sandra Abib

Coordenação UAB-UFSCar

Sandra Abib

Coordenadora do Curso de Sistemas de Informação

Marilde Santos

UAB-UFSCar

Universidade Federal de São Carlos

Rodovia Washington Luís, km 235

13565-905 - São Carlos, SP, Brasil

Telefax (16) 3351-8420

www.uab.ufscar.br

uab@ufscar.br

Sergio Donizetti Zorzo
Rafael Tomé de Souza
Ânderson Kanegae Soares Rocha

Desenvolvimento de Aplicações Web em Java

São Carlos

2016

© 2016, dos autores

Supervisão

Douglas Henrique Perez Pino

Revisão Linguística

Clarissa Galvão Bengtson

Daniel William Ferreira de Camargo

Diagramação

Izís Cavalcanti

Capa e Projeto Gráfico

Luís Gustavo Sousa Sguissardi

SUMÁRIO

APRESENTAÇÃO	7
UNIDADE 1: Internet	11
UNIDADE 2: HTML5 / CSS / JavaScript	23
UNIDADE 3: Servlet	105
UNIDADE 4: JSP	131
UNIDADE 5: Banco de Dados	157

APRESENTAÇÃO

Este texto visa dar apoio ao estudante da disciplina Desenvolvimento de Software para Web do Curso de Bacharelado em Sistemas de Informação da Universidade Federal de São Carlos, oferecido na modalidade a distância.

O curso está estruturado em 5 unidades com o objetivo de capacitar o estudante na construção de um sistema de software para web, utilizando-se de tecnologias para a construção das interfaces do sistema de software e tratamento das requisições no servidor web, com acessos a dados armazenados em banco de dados.

A unidade 1 apresenta a evolução da Internet e das redes de computadores, abordando os conceitos de protocolos responsáveis por permitir a interconexão de máquinas através de uma rede, as principais tecnologias envolvidas nesse processo, a comunicação entre cliente e servidor, e finalizando com a conceituação da World Wide Web (WWW) e os navegadores de Internet.

A unidade 2 apresenta os conceitos do HTML, a linguagem de marcação utilizada para a construção de documentos de página Web. Esta unidade apresenta também o uso de formulários que permitem a interatividade entre o usuário e o servidor em uma particular aplicação Web, coletando dados fornecidos pelo usuário. Tais transações possibilitam a leitura e gravação de informações em bancos de dados pelos servidores, gerando enormes possibilidades de uso da Internet. Também são abordados nesta unidade os conceitos de páginas de estilo (CSS), possibilitando um controle absoluto da aparência da interface Web, e o uso da linguagem de script JavaScript, que possibilita manipular os elementos de marcação do HTML, alterar o estilo de uma determinada TAG ou fazer validações.

A unidade 3 objetiva a apresentação dos Servlets, que são componentes hospedados no servidor e que geram dados HTML e XML para a camada de apresentação de um aplicativo Web. Sendo assim, os assuntos abordados referem-se à estrutura básica do Servlet e seus principais métodos. Um exemplo completo é apresentado explorando as funcionalidades usando a IDE NetBeans.

A unidade 4 apresenta a tecnologia JSP utilizada no desenvolvimento de aplicações para Web e que permite o processamento de linguagens de script no lado servidor para geração de conteúdo dinâmico. O funcionamento do JSP, sua estrutura e os componentes que podem ser empregados na linguagem são apresentados. Para complementar o estudo da tecnologia são apresentados dois exemplos completos de páginas dinâmicas utilizando a tecnologia JSP desenvolvidos na IDE NetBeans.

A unidade 5 objetiva apresentar a ligação de uma página Web com o Banco de Dados MySQL, através da linguagem Java para Web. Exemplo de utilização de um sistema de *software* para web é apresentado como um estudo de caso para a fixação do conteúdo.

Durante todo o texto é realizado o desenvolvimento de um sistema de *software* para web que objetiva exemplificar o uso das tecnologias apresentadas. No texto é apresentado o Sistema de Gerenciamento de Notícias, que compreende um cadastro completo para a criação, alteração, exclusão e visualização de notícias.

Um outro exemplo de sistema de *software* será desenvolvido pelo próprio aluno durante a realização da disciplina, com apoio do professor e dos tutores

Esse texto foi concebido originalmente pelo Prof. Sergio Donizetti Zorzo e teve contribuições dos tutores Paulo Muniz de Ávila, Elaine Cecília Gatto, Anderson Kanegae Soares Rocha e Rafael Tomé de Souza.

UNIDADE 1

Internet

Internet

A palavra Internet é conhecida pela grande maioria das pessoas que vivem nas cidades, mas se perguntarmos o que ela significa, provavelmente ouviremos algo como: “é a grande rede mundial de computadores”. Esta definição não é exata! A Internet poderia ser melhor definida como o conjunto de diversas redes de computadores que se comunicam através dos protocolos TCP/IP.

A Internet vem se apresentando como uma revolução ímpar no mundo dos computadores e das comunicações. Ela representa a integração de várias invenções como o telégrafo, o telefone, o rádio, o computador e até mesmo a televisão. Através desta integração ela se torna um mecanismo de disseminação da informação e divulgação mundial e um meio para colaboração e interação entre indivíduos e seus computadores, independentemente de suas localizações geográficas.

Historicamente, quatro aspectos se destacam sobre a Internet:

- Iniciou com a ARPANET – projeto coordenado pela Agência de Pesquisa Avançada dos Estados Unidos (*Advanced Research Projects Agency – ARPA*) e suas tecnologias;
- A delimitação dos aspectos operacionais e gerenciais de uma infraestrutura operacional complexa e global;
- O aspecto social que resultou numa larga comunidade de internautas trabalhando juntos para criar e evoluir com a tecnologia;
- E o aspecto de comercialização que resulta numa transição extremamente efetiva da pesquisa numa infraestrutura de informação disponível e utilizável.

O termo Internet surgiu em 1983, quando a ARPANET foi dividida em duas redes, uma militar, a MILNET, e uma versão reduzida da ARPANET. A Internet hoje é uma grande coleção de redes interligadas no mundo inteiro, formando uma grande rede de computadores.

Em 24 de outubro de 1995, o Conselho Federal Norte-Americano de Redes (*Federal Networking Council*) aprovou uma resolução definindo o termo Internet, que diz o seguinte:

Internet se refere ao sistema de informação global que -- (i) é logicamente ligado por um endereço único global baseado no Protocolo de Internet (Internet Protocol-IP) ou suas subsequentes extensões; (ii) é capaz de suportar comunicações usando o Protocolo de Controle de Transmissão (Transmission Control Protocol/Internet Protocol-TCP/IP) ou suas subsequentes extensões e/ou outros protocolos compatíveis ao IP; e (iii) provê, usa ou torna acessível,

tanto publicamente como privadamente, serviços de mais alto nível produzidos na infra-estrutura descrita.

1.1 Primeiras Palavras

Essa unidade objetiva apresentar a evolução da Internet e das redes de computadores. Para isso, serão abordados assuntos referentes aos protocolos responsáveis por permitir a interconexão de máquinas através de uma rede, as principais tecnologias envolvidas nesse processo, a comunicação entre cliente e servidor, concluindo a unidade com uma apresentação sobre o World Wide Web (WWW) e os navegadores de Internet.

1.1.1 A Evolução da Internet

O uso da Internet sempre foi exponencial, a quantidade de componentes ou nós (*hosts*) conectados nos seus primeiros dias, cerca de 10 computadores foi para mais de 100 milhões de nós em menos de 30 anos, e continua crescendo. De acordo com dados do Consórcio de Sistemas de Internet Mundial (*Internet Systems Consortium, Inc.*) (ISC 2013) pode-se verificar o crescimento no número de computadores interligados. A tabela a seguir apresenta os dados do número de computadores interligados na Internet no período de 1999 a julho de 2013.

Ano	Mês	Número de Computadores Interligados
1999	Janeiro	43.230.000
	Julho	56.218.000
2000	Janeiro	72.398.092
	Julho	93.047.785
2001	Janeiro	109.574.429
	Julho	125.888.197
2002	Janeiro	147.244.723
	Julho	162.128.493
2003	Janeiro	171.638.297
2004	Janeiro	233.101.481
	Julho	285.139.107
2005	Janeiro	317.646.084
	Julho	353.284.187
2006	Janeiro	394.991.609
	Julho	439.286.364
2007	Janeiro	433.193.199
	Julho	489.774.269

Ano	Mês	Número de Computadores Interligados
2008	Janeiro	541.677.360
	Julho	570.937.778
2009	Janeiro	625.226.456
	Julho	681.064.561
2010	Janeiro	732.740.444
	Julho	768.913.036
2011	Janeiro	818.374.269
	Julho	849.869.781
2012	Janeiro	888.239.420
	Julho	908.585.739
2013	Janeiro	963.518.598
	Julho	996.230.757

O Brasil ocupa a quarta posição em relação ao número total de domínios – que é a identificação de um conjunto de computadores na Internet - conforme ilustrado na tabela a seguir (ISC 2013).

País	Total de Domínios
Estados Unidos (.edu, .us, .mil, .org, .gov, .com, .net e .info)	552.032.998
Japão (.jp)	74.461.142
Alemanha (.de)	34.904.481
Brasil (.br)	33.691.951
Itália (.it)	26.136.473
China (.cn)	19.976.554
México (.mx)	17.658.991
França (.fr)	17.437.386
Austrália (.au)	16.900.586
Rússia (.ru)	15.122.103

De acordo com o relatório de pesquisa sobre o uso das tecnologias da informação e da comunicação no Brasil em 2012, elaborado pelo Centro de Estudos sobre as Tecnologias da Informação e da Comunicação (CETIC.br 2013), pode-se destacar alguns números:

- 46% dos domicílios brasileiros possuem computadores;
- A Internet atingiu 40% do total de domicílios brasileiros;
- Em 2012, pela primeira vez, a proporção de usuários de Internet (49%, aqueles que usaram a Internet há menos de 3 meses) é maior do que a de indivíduos que nunca usaram a Internet (45%).

Apesar do aumento do número de computadores nos domicílios e do acesso a Internet, 42,8 milhões de pessoas de 45 anos ou mais e 68 milhões de pessoas das classes C e DE ainda não usaram a Internet.

1.1.2 Tecnologias Utilizadas

Os protocolos TCP/IP foram criados juntamente com a Internet, como solução para a interligação dos computadores nessa rede. O sucesso da Internet fez com que o TCP/IP se tornasse um padrão de fato para a interligação de computadores, tanto em redes locais quanto de longa distância e demonstrou também a viabilidade do uso da arquitetura TCP/IP em larga escala. As principais características do protocolo TCP/IP são (COMER 2000):

- **Uso de padrões abertos:** Ele não está preso a nenhuma plataforma de *hardware* ou *software* específica. As especificações dos protocolos TCP/IP estão disponíveis na Internet, através de RFCs (*Request for Comments*), promulgadas pela IETF (*Internet Engineering Task Force*).
- **Independência da tecnologia de rede:** Os protocolos TCP/IP usam comutação de pacotes para a comunicação entre as partes e são capazes de funcionar sobre uma variedade de protocolos das camadas de enlace e física, de forma transparente para as aplicações. Por essa razão, é a solução mais escolhida para a interligação de *hardware* e *software* diversos.
- **Protocolos padronizados:** Os protocolos TCP/IP são padronizados não apenas no nível de rede e de transporte, mas também em nível de aplicação. Existem, por exemplo, padrões para *e-mail*, *login* remoto e transferência de arquivos, o que torna mais fácil a confecção de novas aplicações.
- **Interconexão total:** Cada dispositivo conectado a uma rede TCP/IP recebe um endereço único que o identifica como um nó e permite que este nó se comunique e seja acessível de qualquer outro ponto da rede.
- **Confirmações fim-a-fim:** O protocolo TCP/IP fornece um mecanismo de confirmação (*acknowledgement*) entre a origem e o destino final de uma comunicação, em vez de simplesmente entre pares de máquinas ao longo do caminho, o que torna a entrega de dados, como um todo, mais confiável.

A organização dos protocolos de comunicação é feita em camadas, onde cada camada é responsável por funções específicas e com definição de um conjunto de protocolos para esse fim. A proposta mais conhecida para a organização de *software* de comunicação é o modelo OSI (*Reference Model of Open Systems Interconnection*) da ISO (*International Organization for Standardization*).

A arquitetura TCP/IP, assim como OSI realiza a divisão de funções do sistema de comunicação em estruturas de camadas, ilustradas na Figura 1 apresentada a seguir (SOARES, LEMOS e COLCHER 1995) com as seguintes atribuições:

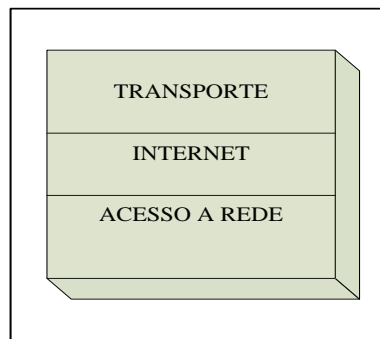


Figura 1 Sistema de comunicação em estrutura de camadas

- **Camada de acesso à rede:** camada de nível mais baixo da hierarquia e fornece a base para a entrega dos dados entre computadores, fazendo referência à placa de rede do computador.
- **Camada Internet:** camada que realiza a comunicação entre máquinas através do protocolo IP. Para identificar cada máquina e a própria rede em que estas estão situadas, é definido um identificador, chamado endereço IP, que é independente de outras formas de endereçamento que possam existir nos níveis inferiores.
- **Camada de Transporte:** camada que reúne os protocolos que realizam as funções de transporte de dados fim-a-fim entre dois nós. A camada de transporte possui dois protocolos que são o UDP e o TCP.
- **Camada de Aplicação:** camada que reúne os protocolos que fornecem serviços de comunicação ao sistema ou ao usuário. Na camada mais alta da arquitetura TCP/IP, encontram-se todos os processos de aplicação que se utilizam dos serviços da camada de transporte para a entrega dos dados. Esses processos optam por um serviço não orientado à conexão, UDP, ou por um serviço orientado à conexão, TCP. Podem-se separar os protocolos de aplicação em duas classes:
 - Protocolos de serviços básicos, que fornecem serviços para atender as necessidades do sistema de comunicação TCP/IP: DNS, BOOTP, DHCP.
 - Protocolos de serviços para o usuário: FTP, HTTP, Telnet, SMTP, POP3, IMAP, NFS, NIS, LPR, ICQ, RealAudio, Gopher, Archie, Finger, SNMP e outros.

O Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol* – HTTP) é um protocolo usado para a transferência de informações na *Web*. Os

dados transferidos pelo protocolo HTTP podem ser hipertextos, textos não estruturados, imagens, vídeo, ou qualquer outro tipo de informação. O HTTP define uma única interação de requisição/resposta, que é vista como uma “transação *Web*”. Cada transação HTTP consiste em uma requisição enviada do cliente ao servidor, seguido de uma resposta enviada do servidor ao cliente, como ilustrado na **Figura 2** (BERNERS-LEE, FIELDING and FRYSTYK 1992).

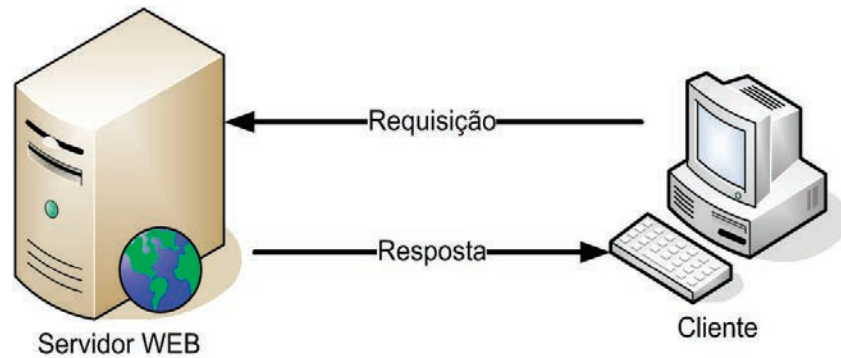


Figura 2 - Transação HTTP

Quando um servidor recebe uma requisição, ele analisa a requisição e verifica qual ação deve ser executada de acordo com o método especificado. O protocolo HTTP define um conjunto de métodos que o cliente pode invocar. Esses métodos funcionam como comandos enviados ao servidor. O protocolo HTTP 1.0 define os métodos descritos na tabela a seguir e o HTTP1.1 acrescenta os métodos definidos na tabela posterior.

Método http 1.0	Descrição
GET	Solicita que seja retornado o recurso identificado pelo Localizador de Recurso Uniforme (<i>Uniform Resource Locator – URL</i>).
HEAD	Obtém informações sobre o recurso sem que o mesmo seja retornado ao cliente. Testa validade de <i>links</i> , acessibilidade e a data da última modificação.
POST	Envia informações adicionais do cliente para o servidor no corpo da mensagem, por exemplo, dados digitados em formulários <i>HTML</i> .

Método http 1.1	Descrição
<i>CONNECT</i>	Reservado para comunicação com servidores <i>Proxy</i> .
DELETE	Solicita que o recurso identificado pela <i>URL</i> seja apagado do servidor.
OPTIONS	Obtém as opções de comunicação disponíveis ou os requisitos associados ao recurso solicitado, sem necessariamente iniciar sua recuperação.
PUT	Permite criar ou modificar um recurso no servidor <i>Web</i> .
TRACE	É utilizado para enviar uma mensagem de teste, do tipo <i>loopback</i> , ao servidor.

As requisições HTTP sempre retornam um código de estado (*Status*), que informa o resultado de sua execução. Esses códigos são divididos nas classes apresentadas a seguir.

Classe	Categoria
100-199	Informativa
200-299	Sucesso
300-399	Redirecionamento
400-499	Erro de Cliente
500-599	Erro de Servidor

O protocolo HTTP não se preocupa com os detalhes de uma comunicação da rede, função desempenhada pelo TCP/IP. As principais etapas envolvidas em uma transação entre o cliente e o servidor são:

- Mapear o nome do servidor para um endereço *IP*;
- Estabelecer uma conexão *TCP* com o servidor;
- Transmitir a requisição com todas as informações necessárias;
- Receber a resposta do servidor;
- Fechar a conexão *TCP*

1.1.3 Conexão

O hardware para conexão à Internet depende do tipo de conexão disponível, que pode ser:

- **Conexão discada:** é preciso ter um *modem* para comunicação com o provedor de acesso via linha telefônica comum.
- **Conexão dedicada ADSL:** é preciso ter uma placa de rede Ethernet 10/100 e um modem ADSL (*Asymmetric Digital Subscriber Line*), além de um separador de sinais do telefone e da transmissão de dados.
- **Conexão dedicada a cabo:** é preciso um *cablemodem* e também um separador de sinais de TV e dos dados.
- **Conexão dedicada wireless:** é preciso um receptor de microondas e uma antena externa para o acesso à rede do provedor.

Já o *software* para efetivar a conexão à Internet pode ser pelo emprego direto do protocolo TCP/IP, o uso de um navegador como o Internet Explorer,

Mozilla Firefox, Google Chrome, Safari, ou um outro *software* específico a depender das necessidades do usuário.

1.1.4 World Wide Web

A *World Wide Web* (também chamada de *Web* ou *WWW*) é, em termos gerais, a interface gráfica da Internet. Ela é um sistema de informação organizado de maneira a englobar todos os outros sistemas de informação disponíveis na Internet.

O princípio básico é criar um mundo de informações sem fronteiras, prevendo as seguintes características:

- Interface consistente;
- Incorporação de um vasto conjunto de tecnologias e tipos de documentos;
- “Leitura universal”, onde todas as pessoas possam utilizar navegadores *Web* com capacidade de interpretar as informações disponíveis na rede.
- Para isso, implementa três ferramentas importantes:
 - Um protocolo de transmissão de dados – HTTP;
 - Um sistema de endereçamento próprio – URL: O sistema de endereçamento da *Web* é baseado em uma sintaxe chamada URI (*Universal Resource Identifier* - Identificador Universal de Recursos). Os endereços que utilizamos atualmente são os URLs (*Uniform Resource Locator* - Localizador Uniforme de Recursos), que seguem essa sintaxe. Um exemplo de URL é: <http://www.dc.ufscar.br/teste/index.php>. Esse endereço identifica:
 - o protocolo de acesso ao recurso desejado (http)
 - a máquina a ser contactada, domínio (www.dc.ufscar.br)
 - o caminho de diretórios até o recurso (teste/), e
 - o recurso (arquivo) a ser obtido (index.php).
 - Uma linguagem de marcação para transmitir documentos formatados através da rede – HTML. A Linguagem de Marcação de Hipertexto – HTML (*HyperText Markup Language*) será apresentada em nossa próxima unidade.

Os navegadores *Web* ou *browsers* são os responsáveis por identificar as marcações em HTML e apresentar os documentos conforme o que foi especificado por essas marcações.

1.2 Considerações finais

Esse capítulo apresentou uma visão geral de redes de computadores e principalmente da Internet e sua relevância no cenário atual.

1.3 Estudos complementares

BERNERS-LEE, T., R. FIELDING, and H. FRYSTYK. *Hypertext Transfer Protocol HTTP/1.0*. RFC, IETF , 1992.

CETIC.br. "Pesquisa sobre o uso das Tecnologias da Informação e da Comunicação no Brasil." *Centro de Estudos sobre as Tecnologias da Informação e da Comunicação*. 2013. <http://www.cetic.br> (acesso em novembro de 2013).

COMER, D. E. *Internetworking with TCP/IP: Principles, Protocols and Architecture*. Prentice Hal, 2000.

ISC. "Internet Systems Consortium, Inc. ." *Internet Systems Consortium, Inc.* julho de 2013. <http://ftp.isc.org/www/survey/reports/2013/07/> (acesso em novembro de 2013).

SOARES, L.F.G., G. LEMOS, e S. COLCHER. *Redes de Computadores. Das LANs, MANs e WANs às Redes ATM*. Editora Campus, 1995.

UNIDADE 2

HTML5 / CSS / JavaScript

HTML5 / CSS / JavaScript

A linguagem padrão utilizada para a criação de documentos na Web é a Linguagem de Marcação de Hipertexto, conhecida como HTML.

O HTML é usado para criação de documentos estruturados de conteúdo a ser manipulado pelos navegadores.

Com a linguagem de marcação de hipertexto HTML é possível criar documentos composto por títulos, parágrafos, imagens, links, campos de formulários e outros elementos que permitem a interação do usuário em uma página Web. O código é armazenado em um arquivo texto, no formato ASCII.

A transferência de um arquivo HTML ocorre via protocolo HTTP. Por isso é comum aparecer no endereço de um site a especificação: `<http://www.dc.ufscar.br>`. Mas os navegadores não exigem a digitação "http://" porque o sufixo é adicionado automaticamente pelo navegador quando o protocolo é usado.

A linguagem de marcação HTML é derivada da Linguagem Padrão de Marcações Genéricas SGML (*Standard Generalized Markup Language*) - ISO 8879, publicado em 1986. A SGML define as regras gerais para uma linguagem de marcação e as linguagens oriundas deste padrão especificam as regras de marcação para uma determinada aplicação, como é o caso de HTML para a exibição de informações na Web.

Outra linguagem de marcação conhecida é o XML, as siglas significam linguagem extensível de marcação (*eXtensible Markup Language*). O XML tem uma aplicação mais ampla em relação a criação de documentos hipertexto proposto pelo HTML que é mais direcionado aos navegadores.

O XML permite que uma organização/entidade crie seus próprios elementos e regras de sintaxe para um determinado propósito. Essa linguagem permite criar documentos direcionados para *softwares* facilitando a troca de dados, garantindo a portabilidade e facilidade na integração. Tudo isso graças a possibilidade de fazer mapeamentos mais flexíveis. Observe o exemplo a seguir que descreve dados de uma pessoa estruturado em um arquivo XML:

CÓDIGO 2.1 - Dados de uma pessoa estruturado em um arquivo XML

```
<?xml version="1.0" encoding="UTF-8"?>
< Pessoa >
  < nome >Joaquim</ nome >
  < profissao >Estagiário</ profissao >
  < endereco >
    < rua >Rua De la Penha</ rua >
    < numero >330</ numero >
    < bairro >Jd. Noruega</ bairro >
    < cep >16530-540</ cep >
  </ endereco >
</ Pessoa >
```

O arquivo HTML pode ser gerado através de qualquer editor de texto, desde que se salve o documento com a extensão “.html” ou “.htm”. Editores de texto tais como o Microsoft Office Word® possuem recursos para salvar documento ou página em HTML.

Além do Microsoft Office Word®, existem outros softwares bem mais simples capazes de criar documentos HTML, tais como: NotePad® - editor de bloco de notas da própria Microsoft®. Ou editores de texto não proprietários tais como Sublime¹ e Notepad++², ambos com recursos adicionais que possuem atalhos e facilitadores para criação de código HTML.

Para exibir um documento HTML é necessário ter instalado um navegador Web (*browser*) em seu computador. Os navegadores mais comuns são o Internet Explorer³ (Microsoft), Chrome⁴ (Google) e o Firefox⁵ (Mozilla).

Para entender um conteúdo de um arquivo HTML é necessário entender os elementos de marcação usados para rotular uma determinada informação.

A informação é marcada por um par de TAGS, etiquetas ou rótulos, inserido no início e no final do conteúdo a ser inserido, veja o exemplo apresentado na Figura 3:

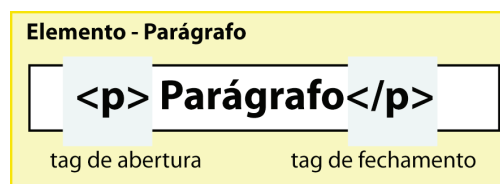


Figura 3 – Definindo marcação de um elemento HTML

- 1 <http://www.sublimetext.com/>
- 2 <http://notepad-plus-plus.org/>
- 3 <http://windows.microsoft.com/pt-br/internet-explorer/download-ie>
- 4 <http://www.google.com/intl/pt-BR/chrome/>
- 5 <http://www.mozilla.org/pt-BR/>

As TAGs aparecem sempre entre os sinais de “menor que” (<) e “maior que” (>). Geralmente são utilizados em pares, sendo que a TAG de finalização de um comando é precedida de uma barra (/).

A palavra escrita na TAG serve para o navegador identificar o tipo de informação que está colocada entre as TAGs de início e fim. Repare que a diferença entre a TAG de início e fim é o uso da barra (/).

Algumas recomendações devem ser observadas ao utilizar as TAGs para realizar marcações:

- **NUNCA** esquecer um sinal de “<” ou de “>”, no início ou no fim da TAG.
- **Não usar espaços na tag**, exemplo: </ p>.

Ignorar essas regras podem gerar problemas ao visualizar a página HTML no navegador. A Figura 4 ilustra um código aplicando essas recomendações e o outro código ignorando tais regras.

```
01. <html>
02. <head>
03.     <title>HTML 01</title>
04. </head>
05. <body>
06.     DSW
07.     <p> Teste </p>
08.     Teste
09. </body>
10. </html>

01. <html>
02. <head>
03.     <title>HTML 02</title>
04. </head>
05. <body>
06.     DSW
07.     p> Teste </ p>
08.     Teste
09. </ body>
10. </html>
```

Figura 4 – Aplicando regras para formatar o código

Ao visualizar a página HTML com o título “HTML 01” pode-se perceber que existem palavras “Teste” uma embaixo da outra.

Um cenário diferente acontece ao visualizar a página HTML com o título “HTML 02”. A palavra “Teste” encontra-se uma do lado da outra. Isso mostra

que ignorar as regras mencionadas fere o comportamento de bloco esperado ao usar a respectiva TAG parágrafo (<p>...</p>).

2.1 Primeiras Palavras

O objetivo principal dessa unidade é apresentar os conceitos do HTML, a linguagem de marcação utilizada para construção de documentos de página Web.

Para complementar o aprendizado, a unidade também apresenta o uso de formulários que permitem a interatividade entre o usuário e o Servidor em uma particular aplicação Web, coletando dados fornecidos pelo usuário.

Finalmente, a unidade apresenta os conceitos de CSS que possibilita um controle absoluto do layout de páginas Web e o uso da linguagem de script JavaScript que possibilita manipular os elementos de marcação do HTML, alterar o estilo de uma determinada TAG e pode ser utilizada para fazer validações.

2.2 Estrutura básica de um documento HTML

A estrutura básica de documento HTML é definido por alguns elementos básicos representados pelas tags `doctype`, `html`, `head` e `body`, como apresentado na Tabela 1.

Tabela 1 Estrutura básica do documento HTML.

<code><!DOCTYPE ...</code>	Existem vários tipos de DOCTYPE que representam versões do HTML que são utilizadas para construção de páginas web. Adiante neste texto são apresentadas possíveis versões e quais serão usadas.
<code><html></code>	Início do documento HTML
<code><head> <title> Título </title> </head></code>	Entre as tags <code><head> . . . </head></code> são armazenadas informações como título, meta tags, inserção de chamadas para scripts usando a linguagem JavaScript, inserção de chamadas de folhas de estilo externa e entre outras informações.
<code><body></code>	Início do corpo do documento
<code><p>Conteúdo</p></code>	Parágrafo a ser exibido.
<code></body></code>	Final do corpo do documento
<code></html></code>	Fim do documento HTML

A Figura 5 apresenta a forma como o navegador Firefox renderiza o código HTML salvo em um arquivo com extensão `“.html”`. Observe que as tags de abertura e fechamento `<title></title>` definem o título da página Web, enquanto as tags de abertura e fechamento `<p></p>` definem o corpo de uma página html.

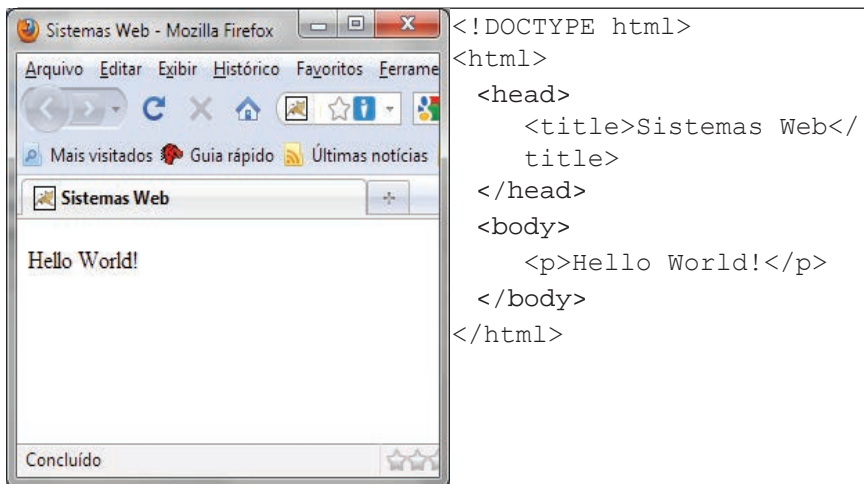


Figura 5 Visualização de página HTML

O documento HTML pode ser criado em *softwares* de editor de texto tais como NotePad®, WordPAD®, Microsoft Office Word®, DreamWeaver®⁶, Sublime⁷, Notepad++⁸ ou qualquer outro editor de sua preferência.

É importante prestar atenção as seguintes regras no momento de salvar um documento com código HTML :

- Salvar o documento com extensões “.html” ou “.htm” para que os navegadores possam reconhecer o arquivo como sendo html. Já para os editores específicos de páginas html o uso da extensão “.html” é automática.
- Os nomes dos arquivos devem iniciar obrigatoriamente com uma letra e podem fazer uso de caracteres como “-” e “_”. Não utilizar caracteres especiais, como: ~, ', , ^, ç, entre outros, pois alguns servidores e navegadores poderão ter dificuldade em interpretá-los.
- Procure usar nomes significativos que remetam ao conteúdo do arquivo.

É importante saber que a linguagem HTML não é do tipo *case sensitive*. Ou seja, não há diferença entre tags de marcação escritas com letras maiúsculas em relação a tags escritas com letras minúsculas, por exemplo <BODY>, <Body>, <body>, <html>, </HTMl> e etc.

A importância em manter um código legível torna necessário utilizar uma única opção – tudo maiúsculo ou tudo minúsculo.

A escolha do tipo de caixa do texto (maiúsculo e minúsculo) faz diferença em algumas versões da HTML, assim como existem versões da HTML que nem todas as tags de abertura exigem tags de fechamento e versões que pede-se tudo que abre deve ser fechado.

6 <https://creative.adobe.com/products/dreamweaver>

7 <http://www.sublimetext.com/>

8 <http://notepad-plus-plus.org/>

A influência direta dessas escolhas nem sempre altera a forma como o seu navegador exibe a página. No entanto, fere os princípios norteados pela W3C⁹, responsável por guiar o desenvolvimento de padrões da Web, assim como validadores de páginas Web que verificam se sua página está de acordo com os padrões da W3C.

2.2.1 HTML e suas versões

O HTML vem sendo evoluído desde a invenção da Web por Tim Berners-Lee em 1992 até os dias da escrita deste documento (2013).

O HTML teve várias versões: HTML, HTML+, HTML 2.0, HTML 3.0, HTML 3.2, HTML 4.0, HTML 4.01 e o HTML5, versão mais recente em fase de adoção pela maioria dos navegadores. Provavelmente, o HTML5 em 2014 passa a ser uma recomendação oficial¹⁰.

Ainda existe o XHTML, recomendação independente da W3C, que é uma reformulação do HTML 4.01 com a aplicação do XML 1.0¹¹ lançado por volta do ano 2000. Sendo assim, a W3C recomenda para criação de páginas e aplicações Web as versões do XHTML 1.1, XHTML 1.0 e o HTML 4.01¹⁰.

XHTML e HTML são linguagens de marcação, entretanto o HTML aceita tanto grafia com letras minúscula ou maiúscula.

Outra característica é que o XHTML tem uma regra de sintaxe mais rígida, por exemplo: a tag que define quebra de linha é escrita como sendo `
` em HTML, mas em XHTML deve ser inserido a barra para fechar a tag `
`.

Essas convenções são mais perceptíveis quando aplica o validador da W3C (<http://validator.w3.org/>) que verifica se o código está seguindo as diretrizes de cada linguagem.

A versão mais nova do HTML é denominada HTML5 e também pode ser usada. No entanto, deve ser tomado cuidado com o uso de algumas funcionalidades, pois nem todos os navegadores apresentam suporte completo aos recursos trazidos pelo HTML5.

As versões do HTML são determinadas por uso de DOCTYPEs específicos logo no início de sua página HTML como apresentado na **Tabela 1**, estrutura básica de um documento HTML.

9 <<http://www.w3c.br/Sobre>>

10 FLATSCHART, Fábio - HTML 5 - Embarque Imediato, Rio de Janeiro, Brasport, 2011.

11 SILVA, Maurício S. – Criando sites com HTML: Sites de alta qualidade com HTML e CSS, São Paulo, Novatec Editora, 2008.

2.2.2 DOCTYPE

Para identificar a versão do HTML que uma página Web está utilizando basta analisar a primeira linha do código HTML. A primeira linha possui uma tag que tem a função de informar ao navegador como o documento precisa ser processado.

A tag é conhecida como DOCTYPE e indica que um documento é escrito na linguagem HTML usando regras de um DTD (*Declaração do tipo de documento*) publicado no site da W3C. Isso considerando que a versão do HTML usada seja 4.01 ou XHTML 1.0 ou 1.1, veja nos exemplos a seguir:

HTML 4.01 Strict

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

HTML 4.01 Transitional

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
```

HTML 4.01 Frameset

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
  "http://www.w3.org/TR/html4/frameset.dtd">
```

XHTML 1 Strict

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

XHTML 1 Transational

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transational//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transational.dtd">
```

XHTML 1 Frameset

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

Em cada um dos DOCTYPEs é possível analisar a presença de palavras em inglês tais como *strict*, *transational* e *frameset*. Essas palavras indicam se os DOCTYPEs suportam atributos ou elementos em desuso pela W3C, assim como apresenta ou não suporte de recursos de frame.

Pode se entender que essas palavras representam o tipo de DOCTYPE. Assim, o tipo ***strict*** não aceita elementos ou atributos em desuso e nem atributos ou elementos destinado a marcação de frameset. Ao contrário, do DOCTYPE do tipo ***transational*** que apresenta suporte aos elementos em desuso pela W3C, mas não aceita elementos ou atributos para marcação de frameset.

Por fim, o tipo de DOCTYPE *frameset* aceita tanto elementos e atributos em desuso e aqueles para marcação de frames.

Os anos passaram e o DOCTYPE não sofreu tantas alterações nas suas diversas versões. No entanto, a versão do HTML5 mudou radicalmente o DOCTYPE tirando a necessidade de informar o DTD a ser usado. Consequentemente, ficou mais fácil de lembrar a sintaxe da primeira linha como pode ser visualizado no exemplo a seguir:

```
<!DOCTYPE html >
```

Todos os exemplos, assim como a aplicação Web a ser desenvolvida no decorrer do livro usará o DOCTYPE do HTML5. Entende-se que HTML5 é a linguagem mais recente a ser usada para construção de código HTML.

No entanto, o enfoque que o livro dará é em relação as principais tags do HTML que são válidas também em versões anteriores da linguagem, com exceção de alguns atributos que funcionam apenas no HTML5.

As TAGs a serem apresentadas auxiliarão na criação de layouts e formulários para suas aplicações. As outras tecnologias que circundam o mundo do HTML5 não serão foco desse livro.

2.3 HTML5

Por volta de 2002 a W3C liberou uma versão do XHTML1.1 que era essencialmente XML, enquanto a versão XHTML 1.0 era essencialmente HTML com um pouco de XML. A versão XHTML 1.1 apresentava problemas de compatibilidade com versões de navegadores da época e antigos.¹²

Em 2004, um grupo formado por pessoas da Apple, Mozilla e Opera começaram a trabalhar em uma especificação que garantia a compatibilidade com versões anteriores podendo ser usada para criação de aplicações Web. A W3C rejeitou a proposta e isso conduziu a criação do grupo WHATWG (*Web Hypertext Application Technology Working Group*).

Inicialmente o grupo trabalhou em duas especificações distintas *Web Forms 2.0* e *Web Application 1.0*, tempos depois fundiram-se em uma especificação denominada HTML5.¹²

No ano de 2009 a W3C anunciou que estava parando os trabalhos em relação a especificação do XHTML 2.0 e que seguiria com o HTML5. O grupo

WHATWG continua com o desenvolvimento da especificação, enquanto o trabalho da W3C é focar nas revisões.¹²

A linguagem HTML5 foi construída com o princípio de ter compatibilidade com versões anteriores. Ou seja, o HTML5 tem como base o HTML4 garantindo o correto suporte à linguagem HTML, além dos novos elementos embutidos na linguagem pela versão 5.¹²

A ideia do HTML5 é permitir a codificação com as mesmas regras de sintaxe das versões anteriores (HTML e XHTML), sendo que os navegadores saberão interpretar corretamente a página.¹²

Com a especificação do HTML5 é possível a criação de aplicações Web usando o conjunto de tecnologias que norteiam o termo HTML5. Algumas das principais melhorias trazidas pela linguagem são melhoria da semântica, novos elementos de formulários, *canvas*, recursos *drag and drop*, armazenamento, vídeo, áudio, *websocket* e entre outros recursos.

2.3.1 Tag <html> e sintaxe do código HTML

A tag <html>...</html> **representa a raiz de uma página Web**. Sendo assim, o navegador ao exibir a página organiza a estrutura em um modelo de objeto conhecido como **DOM (Document Object Model)**. Veja o DOM criado ao visualizar a página da **Tabela 1** em um navegador Web como apresentado na Figura 6.

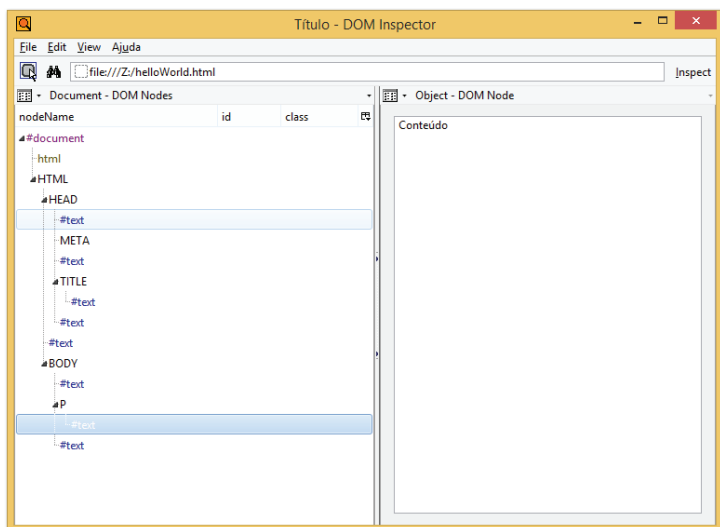


Figura 6 Plugin DOM Inspector p/ Firefox disponível em <https://addons.mozilla.org/pt-BR/firefox/addon/dom-inspector-6622/?src=search>

O elemento HTML tem o atributo `lang` que não é obrigatório, mas auxilia os mecanismos de busca a interpretar qual linguagem é tratada nas páginas, como apresentado a seguir: `<html lang="pt_BR">`.

Dica: o atributo `lang` pode ser incluído em outros elementos, indicando que um determinado parágrafo é de uma citação em outra língua.

Exemplo: `<p lan="en">Hello World!</p>`

Dentre as diversas versões do HTML, existe algumas diferentes maneiras que o código pode ser escrito.

Como o foco do curso é o HTML5 seguem algumas direções de como o código pode ser grafado, de acordo com essa versão segue as seguintes dicas:

- O HTML5 é flexível em relação a escrita, então é possível optar por escrever suas tags em letras maiúsculas ou minúsculas, por exemplo:

```
<p>Trata-se de um parágrafo</p>
```

```
<P>Trata-se de um parágrafo</P>
```

- Alguns elementos tem a tag de fechamento opcional. Destaca-se algumas tags, tais como: `html`, `head`, `body`, `li` e `option`;
- Elementos vazios podem ter a barra para fechar ou não: `<hr>` ou `<hr />`;

Em geral, os desenvolvedores seguem a rigidez da escrita do XHTML. Ou seja, adotando letras minúsculas, usando as tags de abertura e fechamento quando possível, escrevendo os atributos com os valores em aspas duplas ou simplificado apenas quando convém como o exemplo apresentado a seguir:

```
<option selected>
```

```
<option selected="selected">
```

2.3.2 Seção head

O elemento `head` define o cabeçalho de uma página Web e é representado pelo uso das tags `<head> . . . </head>`. O cabeçalho pode conter informações sobre o documento tais como estilo e scripts adicionais.

Além de ter obrigatoriamente o título da página representado pelo elemento `title` (título). Outros elementos podem aparecer no head tais como as metatags.

Uma das metatags opcionais mais comum é a usada para identificar o tipo de caracter utilizado na página Web, como apresentado no CÓDIGO 2.2:

CÓDIGO 2.2 – Código com metatag para identificar tipo de caracter.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Título</title>
  </head>
  <body>...</body>
</html>
```

2.3.3 Seção body

O body refere-se a todo conteúdo visível dentro de uma página HTML. Sendo assim, o conteúdo a ser renderizado pelo navegador deve estar marcado pelo elemento body, representado pelas tags <body> . . . </body> como apresentado no CÓDIGO 2.3:

CÓDIGO 2.3 – Exemplo do uso da tag body

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Título</title>
  </head>
  <body>
    Conteúdo a ser visualizado
  </body>
</html>
```

Alguns dos principais elementos são apresentados no decorrer desta Unidade e, de acordo com a necessidade, outros elementos podem ser apresentados no decorrer do livro.

2.3.4 Lista de elementos

O HTML5 trouxe novos elementos que permitem distinguir em uma página Web onde é o cabeçalho (<header>), rodapé (<footer>) e conteúdo (<aside>, <section> e <article>) principal da página.

Os novos elementos tem uma função semântica que permitem estruturar melhor uma página Web, ajudando a criar o *layout* como apresentado na Figura 7 a seguir.

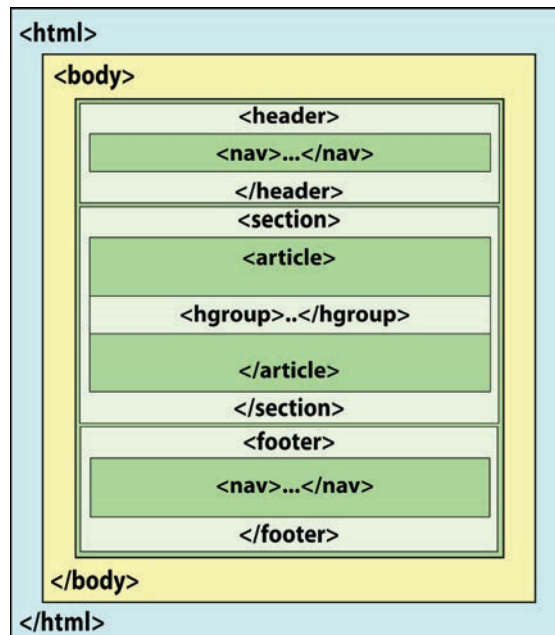


Figura 7 Documento HTML com os novos elementos

Em um futuro, robôs de busca e outros *user-agents* podem ser beneficiados com essas novas marcações habilitadas na linguagem.

A seguir estão relacionados as principais tags que podem ser usadas somente com o HTML5 e outras tags que valem para o HTML5 e versões anteriores;

a) header `<header>...</header>` novo (a partir da versão 5)

Definição: O elemento **header** permite definir um agrupamento de itens relacionados ao cabeçalho de uma página podendo conter navegação, logo, índices de conteúdo, etc.

Suporte: Navegadores mais recentes.

CÓDIGO

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Pipoca</title>
  </head>
  <body>
    <header>
```

```

    
    <nav>
      <ul>
        <li><a href="#">Filme</a></li>
        <li><a href="#">Séries</a></li>
        <li><a href="#">Comente</a></li>
        <li><a href="#">Registra-se</a></li>
      </ul>
    </nav>
  </header>
</body>
</html>

```

b) section <section>...</section> novo (a partir da versão 5)

Definição: O elemento **section** define uma área genérica em uma página Web. A página principal de um website de um cinema pode ser dividida entre destaque do dia, próximos lançamentos e contato. E cada uma dessas área podem ser marcadas como **section**.

Suporte: Navegadores mais recentes.

CÓDIGO

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Pipoca</title>
  </head>
  <body>
    <section>
      <article>
        
      </article>
    </section>
  </body>
</html>

```

c) nav <nav>...</nav> novo (a partir da versão 5)

Definição: O elemento **nav** marca a navegação principal do website. Geralmente a navegação principal está presente no topo da página e/ou no rodapé da página com os links de acesso rápido.

Suporte: Navegadores mais recentes.

CÓDIGO

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Pipoca</title>
  </head>
  <body>
    <header>
      
      <nav>
        <ul>
          <li><a href="#">Filme</a></li>
          <li><a href="#">Séries</a></li>
          <li><a href="#">Comente</a></li>
          <li><a href="#">Registra-se</a></li>
        </ul>
      </nav>
    </header>
  </body>
</html>
```

d) aside <aside>...</aside> novos (a partir da versão 5)

Definição: O elemento **aside** referencia um conteúdo secundário a outro elemento. Pode ser usado para representar conteúdo de publicidade, barras laterais ou para rotular um conteúdo separado do conteúdo principal do site.

Suporte: Navegadores mais recentes.

CÓDIGO

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Pipoca</title>
  </head>
  <body>
    <article>
      <h1>Thor é destaque do ano de 2013!</h1>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      Aenean arcu libero, rutrum ac metus a, aliquet vestibulum
      lacus.</p>
    </article>
    <aside>
      <h2>Notícias relacionadas</h2>
```

```

    <ul>
      <li><a href="#">Homem de Ferro 3</a></li>
      <li><a href="#">Batman fracasso de bilheteria</a></li>
      <li><a href="#">Capitão América não participa da
segunda versão dos Avengers</a></li>
    </ul>
  </aside>
</body>
</html>

```

e) footer <footer>...</footer> novos (a partir da versão 5)

Definição: O elemento **footer** relaciona todo o conteúdo que o desenvolvedor considera como sendo pertencente ao rodapé de um documento HTML.

Suporte: Navegadores mais recentes.

CÓDIGO

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Pipoca</title>
  </head>
  <body>
    <footer>
      <nav>
        <ul>
          <li><a href="#">FAQ</a></li>
          <li><a href="#">Política de privacidade</a></li>
          <li><a href="#">Termos de condição e uso</a></li>
        </ul>
      </nav>
    </footer>
  </body>
</html>

```

f) article <article>...</article> novos (a partir da versão 5)

Definição: O elemento **article** está relacionado a um texto comum dentro de uma página Web, um post ou comentários de usuários.

Suporte: Navegadores mais recentes.

CÓDIGO

```

<!DOCTYPE html>
<html>
  <head>

```

```

    <meta charset="utf-8"/>
    <title>Pipoca</title>
  </head>
<body>
  </header>
  <section>
    <article>
      
    </article>
  </section>
</body>
</html>

```

g) **hgroup** <hgroup>...</hgroup> novos (a partir da versão 5)

Definição: O elemento **hgroup** permite o agrupamento de *headings* quando são apresentados na ideia de título e subtítulo. Logo os *headings* podem variar de h1-h6, representando os diversos níveis de cabeçalho que vão do maior **h1** até o menor **h6**.

Suporte: Navegadores mais recentes.

CÓDIGO

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Pipoca</title>
  </head>
<body>
  <header>
    <hgroup>
      <h1>Pipoca Filmes</h1>
    </hgroup>
  </header>
</body>
</html>

```

h) **divisão** <div>...</div>

Definição: O elemento **div** permite criar blocos genéricos dentro do código HTML. Ou seja, com o elemento **div** o desenvolvedor pode criar seções nível de bloco dentro do código HTML sem valor semântico, como se fosse uma camada.

Seções nível de bloco é quando as tags de abertura e fechamento **div** circundam o conteúdo, gerando um bloco e provocando a quebra de linha do conteúdo antes e depois do conteúdo.

Dentro desse bloco formado pelo elemento `<div>...</div>` é possível usar recursos de estilo (CSS) para formatação ou manipular o **div** com a linguagem de script JavaScript.

CÓDIGO

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>HTML5</title>
  </head>
  <body>
    O <div>desenvolvimento</div> de páginas com HTML.
  </body>
</html>
```

i) paragraph `<p>...</p>`

Definição: O elemento **p** define um bloco de texto dentro de um código HTML, como se fosse um parágrafo e gerando uma seção nível de bloco.

CÓDIGO

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Pipoca</title>
  </head>
  <body>
    <p>
      O desenvolvimento de páginas com HTML.
    </p>
  </body>
</html>
```

j) span `...`

Definição: O elemento **span** define uma seção genérica dentro de um código HTML sem valor semântico. Ou seja, esse elemento gera um bloco no código, nível de linha. O bloco nível de linha faz com que o texto depois da tag de fechamento `` continue na mesma linha. No entanto, um bloco de texto é formado entre as tag de abertura e de fechamento.

A tag **span** pode ser usada juntamente com CSS para formatação de estilo ou manipulado por linguagem de script denominada JavaScript.

CÓDIGO

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>HTML5</title>
</head>
<body>
  O <span>desenvolvimento</span> de páginas com HTML.
</body>
</html>
```

k) lista ordenada ...

Definição: O elemento de lista ordenada cria uma lista no documento HTML aplicando uma numeração sequencial do lado esquerdo da lista.

CÓDIGO

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Lista Ordenada</title>
  </head>
  <body>
    <ol>
      <li>Fluminense</li>
      <li>Corinthians</li>
      <li>Ceará</li>
      <li>Internacional</li>
      <li>Cruzeiro</li>
      <li>Avaré</li>
      <li>Santos</li>
      <li>Botafogo</li>
      <li>Guarani</li>
      <li>Flamengo</li>
      <li>Vasco</li>
      <li>São Paulo</li>
    </ol>
  </body>
</html>
```

Atributos do elemento:

Atributo	type
Valores	1 ou a ou A ou i ou I
Definição e exemplo	O atributo type define o estilo da numeração a esquerda da lista. Exemplo: <ol type="A">...
Atributo	start
Valores	Exemplo : 4
Definição e exemplo	Inicia a lista a partir do número 4. Exemplo: <ol type="1" start="4">...

l) lista desordenada ...

Definição: O elemento de lista desordenada ou não ordenada cria uma lista no documento HTML aplicando símbolos tais como: quadrado, círculo ou disco do lado esquerdo da lista.

CÓDIGO

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Lista Não-Ordenada</title>
  </head>
</html>
<body>
  <ul>
    <li>Palmeiras x Atlético-PR</li>
    <li>Atlético-MG x Guarani</li>
    <li>Flamengo x Ceará</li>
    <li>Atlético-GO x Botafogo</li>
  </ul>
</body>
</html>
```

Atributos do elemento:

Atributo	type
Valores	disc ou square ou circle
Definição e exemplo	É possível mudar o desenho dos marcadores que aparecem do lado esquerdo da lista especificando o type na tag ul, como exemplificado a seguir: <ul type="square">...

m) lista de definição <dl>...</dl>

Definição: O elemento lista de definição é composto pelas tags `dl`, `dt` e `dd`. A tag `dl` é o container que engloba o título da lista marcado pela tag `dt` que por sua vez tem a sua definição marcada pela tag `dd`.

CÓDIGO

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>Lista de definições</title>
</head>
<body>
  <dl>
    <dt>HTML</dt>
    <dd>HiperText Markup Language</dd>
    <dt>OL</dt>
    <dd>Listas Ordenadas - com numeração</dd>
    <dt>UL</dt>
    <dd>Listas sem ordenação</dd>
    <dt>LI</dt>
    <dd>Elemento de lista</dd>
  </dl>
</body>
</html>
```

n) som <audio>...</audio> novos (a partir da versão 5)

Definição: O HTML5 trouxe o elemento **audio** para trabalhar com o som. Essa tag muito menos complexa que as anteriores evita o uso de *plugins* externos em uma página Web tais como *flashplayer* ou *quicktime*. Dessa forma, a responsabilidade de identificar um *player* e *codec* adequado passa para o navegador.

Com o elemento **audio** é possível definir uma única fonte de origem do som ou definir uma sequência de locais onde a fonte de origem de som pode ser localizado.

O formato de áudio tais como **Ogg Vorbis** (*.ogg,*.oga) passa a ser suportado por *Chrome*, *Firefox* e *Opera*. Além deste formato, os formatos mais convencionais continuam a ser suportado tais como **Wav** pelo *Firefox*, *Opera 10.5*, *Safari*, *IE9* e o formato **MP3** pelo *Chrome*, *Safari* e *IE9*.

Suporte: Navegadores mais recentes.

CÓDIGO

```
<!DOCTYPE html>
<html>
<head>
  <title>Audio</title>
</head>
<body>
  <audio controls="controls">
    <source src=" audio/song.ogg" type="audio/ogg"/>
    <source src="audio/song.mp3" type="audio/mp3"/>
    <p>Seu navegador não suporta a tag áudio</p>
  </audio>
</body>
</html>
```

Observação: O navegador busca qual é o formato melhor suportado, através das vários formatos alternativos disponibilizados.

Atributos do elemento:

Atributo	autoplay
Valores	autoplay
Definição e exemplo	Determina se o som deve iniciar automaticamente ou esperar a interação do usuário. Ex: <audio autoplay>...</audio>
Atributo	controls
Valores	controls
Definição e exemplo	Determina se deve aparecer os controles de play, stop, pause e outros. Ex: <audio controls>...</audio>
Atributo	loop
Valores	loop
Definição e exemplo	Determina que o som deve ser reiniciado assim que terminar a música. Ex: <audio controls loop>
Atributo	src
Valores	url
Definição e exemplo	Indica a origem do arquivo a ser reproduzido. Ex: <audio><source src=" "/></audio>

o) vídeo <video>...</video> novos (a partir da versão 5)

Definição: O HTML5 trouxe o elemento **video** que permite inserir vídeos em uma página Web livrando o usuário do uso de *plugins* tais como *quicktime* ou *flashplayer*.

O elemento **video** é mais simplificada que comparada com versões de elementos anteriores para inserção de vídeo na Web. Assim como o elemento **audio**, o elemento **video** tem o recurso de definir fontes alternativas para obter o **video** desejado.

Suporte: Navegadores mais recentes.

CÓDIGO

```
<!DOCTYPE html>
<html>
<head>
  <title>Video</title>
</head>
<body>
  <video controls>
    <source src="video/movie.mp4" type="video/mp4;">
    <source src="video/movie.ogv" type="video/ogg;">
    <p>Seu navegador não suporta a tag vídeo</p>
  </video>
</body>
</html>
```

Atributos do elemento:

Atributo	autoplay
Valores	autoplay
Definição e exemplo	Determina se o video deve iniciar automaticamente ou esperar a interação do usuário. Ex: <video autoplay>...</audio>
Atributo	controls
Valores	controls
Definição e exemplo	Determina se deve aparecer os controles de play, stop, pause e outros. Ex: <video controls>...</audio>
Atributo	loop
Valores	loop
Definição e exemplo	Determina que o video deve ser reiniciado assim que terminar. Ex: <audio controls loop>
Atributo	src
Valores	url
Definição e exemplo	Indica a origem do arquivo a ser reproduzido. Ex: <video><source src="" /></video>

Atributo	width
Valores	Tamanho em pixel
Definição e exemplo	Define em pixels a largura do reprodutor de vídeo Ex: <video width="272"></video>
Atributo	height
Valores	Tamanho em pixel
Definição e exemplo	Define em pixels a altura do reprodutor de vídeo Ex: <video width="272" height="170"></video>

p) âncoras <a>...

Definição: O elemento âncora <a> gera os links de uma página Web. Ou seja, permite que textos, imagens, animação ou qualquer outro objeto possa ser usado para interligar documentos a um outro documento.

O recurso de hipertexto tão importante para a World Wide Web permite referenciar um endereço externo (**url**), uma página do próprio site (**endereço relativo**) ou referenciar um endereço para uma área específica da página Web chamado de **âncora interna**.

CÓDIGO

- Endereço externo:

```
<a href="http://www.dc.ufscar.br">Site</a>
```

- Endereço relativo:

```
<a href="index.html">Home</a>
```

- Email:

```
<a href="contato@dc.ufscar.br">Email</a>
```

- Âncora interna:

```
<!DOCTYPE html>
<html>
<head>
  <title>Âncora Interna</title>
</head>
<body>
  <!--Local que deseja acessar -->
  <a id="acessar">Topo</a>

  <!--conteúdo-->
```

```

<!--Link para acessar o texto topo -->
<a href="#acessar">Voltar</a>
</body>
</html>

```

Atributos do elemento:

Atributo	download <small>(novo)</small>
Valores	Pode ser colocado o nome do arquivo e no atributo href deve ser especificado o caminho para download.
Definição e exemplo	O atributo download força que o href seja feito download. Ex: <pre> </pre>
Atributo	href
Valores	{URL}
Definição e exemplo	Define o caminho para acessar o outro documento HTML. Pode ser uma endereço externo, um endereço relativo ou uma âncora interna.
Atributo	target
Valores	_blank (abre o link em uma nova janela), _parent (abre na janela pai do documento atual) e _self (abre o link na mesma janela)
Definição e exemplo	Determina como o documento do link vai ser aberto. Ex: <pre> Site </pre>

q) imagem

Definição: O elemento **imagem** permite inserir figuras em um documento HTML. Sendo necessário apenas especificar o caminho onde a imagem está salva no servidor.

CÓDIGO

```

 ou


```

Observação: Ao contrário dos outros elementos apresentados, o **img** é um **elemento vazio**. Ou seja, não existe uma tag `` para fechamento. Para encerrar basta usar `>` dentro da mesma tag que define o `src` ou simplesmente não fechar. Já que isso é permitido na versão 5 do HTML.

Atributos do elemento:

Atributo	src
Valores	{caminho da imagem salva}
Definição e exemplo	Especifica o local onde a imagem está localizada podendo incluir diretórios e subdiretórios.
Atributo	alt
Valores	{Inserir um nome alternativo p/ imagem}
Definição e exemplo	Fornece o texto que aparece no espaço da imagem quando o navegador não pode abrir ou após o seu carregamento.
Atributo	width
Valores	Pode ser em pixel ou %
Definição e exemplo	Define a largura da imagem. Sendo a largura padrão o tamanho da própria imagem. Ex: <code><img src="" width="100px"</code>
Atributo	height
Valores	Pode ser em pixel ou %
Definição e exemplo	Define a altura da imagem. Sendo a altura padrão o tamanho da própria imagem. Ex: <code><img src="" height="100px"</code>

Dicas:

- Caso não seja definido os atributos *width* e *height*, a imagem é visualizada de acordo com o seu tamanho real;
- Coloque todas as imagens em um subdiretório específico. Isto ajuda a organizar os arquivos e padronizar o seu projeto Web.

r) tabela `<table>...</table>`

Definição: Os dados podem ser apresentados em uma página Web dentro de tabelas. Para isso, pode ser usado o elemento tabela através das tags de abertura e fechamento `<table>...</table>`. As tabelas não devem ser usadas para controlar o *layout* da página, isso deve ser responsabilidade da folha de estilo (CSS).

CÓDIGO

Entre as tags `<table>...</table>` define-se as linhas `<tr>...</tr>` e as colunas de cada linha `<td>...</td>`. Observe o código a seguir:

```
<!DOCTYPE html>
<html>
<head>
  <title>Tabelas</title>
</head>
<body>
  <table>
    <tr>
      <td>v1</td>
      <td>1</td>
      <td>2</td>
    </tr>
    <tr>
      <td>v2</td>
      <td>3</td>
      <td>4</td>
    </tr>
  </table>
</body>
</html>
```

Ainda existem as tags de abertura e fechamento `<thead>..</thead>`, `<tbody>...</tbody>` e `<tfoot>...</tfoot>` que podem ser usadas para agrupar as células de uma tabela indicando o cabeçalho, o corpo e o rodapé.

O uso dessas tags são úteis para serem usadas juntamente com a folha de estilo, veja o código a seguir:

```
<!DOCTYPE html>
<html>
<head>
  <title>Tabelas</title>
</head>
<body>
  <table>
    <p>Tabela para o churrasco:</p>
    <thead>
      <tr>
        <th>Nome</th>
        <th>Valor</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>Joaquim</td>
```

```

        <td>R$ 50,00</td>
    </tr>
    <tr>
        <td>Antunes</td>
        <td>R$ 50,00</td>
    </tr>
</tbody>
<tfoot>
    <tr>
        <td>Total</td>
        <td>R$ 100,00</td>
    </tr>
</tfoot>
</table>
</body>
</html>

```

Dentre os atributos que não foram descontinuados na versão 5 do HTML, vale a pena verificar o atributo *colspan* e *rowspan* pertencente ao elemento **coluna** `<td>...</td>`.

O atributo *rowspan* e *colspan* pode ser usado para **mesclar linhas (row)** ou **mesclar colunas (col)**, como é o caso do exemplo apresentado na Figura 8.

Contato		
Nome	Telefone	Cidade
Joaquim Felipe	18-99999999	São Carlos/SP
Antunes Almirante	14-99999999	São Carlos/SP
Trovô Almeida	16-99999999	São Carlos/SP

Figura 8 - Tabela usando o atributo *colspan*

A Figura 8 apresenta uma tabela em que a primeira linha da tabela foi mesclada com o atributo *colspan* pertencente a tag `td`. Verifique o código a seguir:

```

<table>
  <thead>
    <tr>
      <th colspan="3" class="head">Contato</th>
    </tr>
    <tr>
      <th>Nome</th>
      <th>Telefone</th>
      <th>Cidade</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Joaquim Felipe</td>
      <td>18-99999999</td>

```

```

        <td>São Carlos/SP</td>
    </tr>
    <tr>
        <td>Antunes Almirante</td>
        <td>14-99999999</td>
        <td>São Carlos/SP</td>
    </tr>
    <tr>
        <td>Trovô Almeida</td>
        <td>16-99999999</td>
        <td>São Carlos/SP</td>
    </tr>
</tbody>
</table>

```

2.3.5 Elementos de formulário

Novos elementos de formulário foram introduzidos com a especificação do HTML5. No entanto, nem todos os elementos possuem pleno suporte dos navegadores mais recentes.

Às vezes existem navegadores com total suporte as especificações do HTML5 e outros que ainda não foram implementados o suporte total a linguagem.

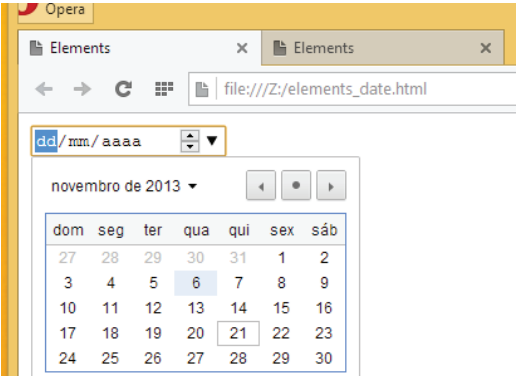
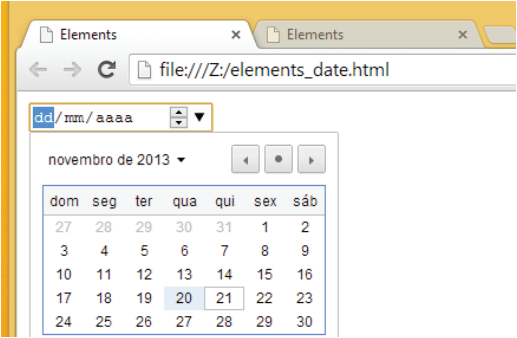
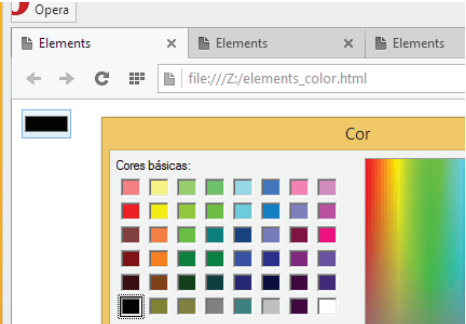
Sendo assim, serão apresentados alguns elementos que podem ser usados no HTML5 e versões anteriores. E quando for apresentado algum elemento novo será indicado qual navegador que o melhor suporta.

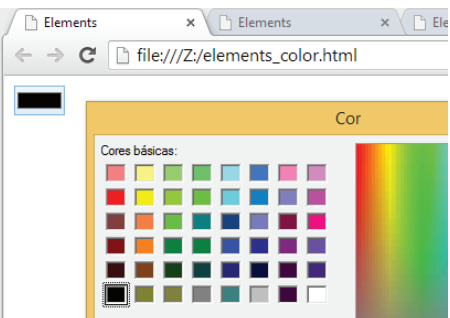
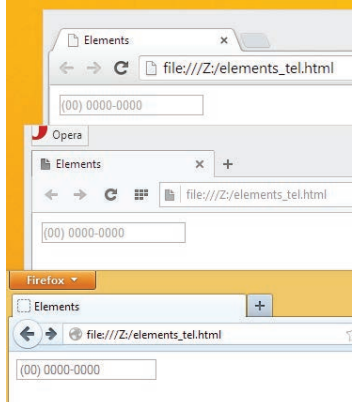
Lembre-se de quando criar a sua aplicação levar isso em consideração e verificar se o elemento a ser utilizado apresenta suporte já implementado pela maioria dos navegadores.

a) **Entrada de texto** <input...>

O elemento **input** é usado para apresentar diversos tipos de objetos no documento HTML tais como campo texto, *checkbox*, campo para seleção e botão que permite que o usuário possa inserir dados e submeter os dados a aplicação.

Atributos do elemento:

Atributo	type
Valores	date (novo) apenas HTML5
Definição	Criar um componente no navegador com o formato de yyyy-mm-dd.
Código	<code><input type=date></code>
Navegador Opera v18.0	
Navegador Chrome v31.0.1650.57 m	
Atributo	type
Valores	color (novo) apenas HTML5
Definição	Criar um componente no navegador para selecionar cores.
Código	<code><input type=color placeholder=black></code> Obs: placeholder é um atributo que apresenta uma dica para o usuário
Navegador Opera v18.0	

Navegador Chrome v31.0.1650.57 m	
Atributo	type
Valores	tel (novo) apenas HTML5
Definição	Criar um componente no navegador para o usuário inserir o número do telefone. Existe um atributo placeholder que é uma dica de como o usuário deve realizar a entrada do dado.
Código	<code><input type="tel" placeholder="(00) 0000-0000"></code>
Navegador Opera v18.0, Chrome v31.0.1650.57 m e Firefox 25.0.1	
Atributo	type
Valores	text
Definição	Permite o usuário inserir texto.
Código	<code><input type="text" name="nome"></code>
Atributo	type
Valores	password
Definição	Definido como campo senha, permite o usuário inserir um texto que é mascarado com asteriscos ou círculos
Código	<code><input type="password" name="senha"></code>
Atributo	type
Valores	checkbox
Definição	Checkbox permite o usuário selecionar mais de uma opção.
Código	<code><input type="checkbox" name="veiculo" value="carro">Carro
<input type="checkbox" name="moto" value="moto">Moto
</code>
Atributo	type
Valores	hidden
Definição	Cria um objeto oculto dentro do HTML.
Código	<code><input type="hidden" name="idProduto"/></code>

Atributo	type
Valores	radio
Definição	Permite o usuário selecionar um valor ou outro.
Código	<code><input type="radio" name="bebida" value="Refrigerante">Refrigerante
<input type="radio" name="bebida" value="Suco">Suco
</code>
Atributo	type
Valores	submit
Definição	O tipo submit habilita a submissão de páginas Web.
Código	<code><input type="submit" value="Enviar"/></code>
Atributo	value
Definição	Atributo válido para o type = text . Armazena o conteúdo inicial do objeto. Dependendo do tipo de objeto (ex.: caixas de texto) o conteúdo pode ser manipulado pelo usuário. Exemplo: <code><input type="text" value="Rio de Janeiro"></code>
Atributo	size
Definição	Atributo válido para o type = text . Indica o número de caracteres visíveis ("tamanho" do campo) de um objeto tipo "TEXT" (caixa de texto). Exemplo: <code><input type="text" size="40"></code>
Atributo	maxlength
Definição	Atributo válido para o type = text . Indica o número máximo de caracteres permitido. Exemplo: <code><input type="text" size="40" maxlength="60"></code>
Atributo	checked
Definição	Atributo válido para o type = radio type = checkbox Determina qual a opção padrão (default) para objetos tipo "RADIO". Para objetos tipo "CHECKBOX" determina qual opção está assinalado. Exemplo: <code><input type="checkbox" checked></code>
Atributo	name
Definição	Indica o nome do objeto a ser usado pelo script associado ao formulário para a manipular dados. Exemplo: <code><input type="text" name="endereco"></code>

b) textarea <textarea>...<textarea>

Definição: Permite digitar múltiplas linhas de texto em um documento HTML.

CÓDIGO

```
<textarea rows="3" cols="50" name="objeto"> Área para texto
</textarea>
```

Atributos do elemento:

Atributo	rows
Definição	Indica o número de linha que a área de texto deve possuir
Atributo	cols
Definição	Indica o número de colunas que a área de texto deve possuir.
Atributo	name
Definição	Indica o nome do objeto a ser usado pelo script associado ao formulário para a manipular dados.

c) select <select>...<select>

Definição: Selecionar uma opção dentre uma lista de opções possíveis.

CÓDIGO

```
<select name="opcoes">
  <option selected> Texto A </option>
  <option> Texto B </option>
</select>
```

Atributos do elemento:

Atributo	value
Definição	Define o valor da opção selecionada.
Atributo	selected
Definição	Define qual opção é selecionada como default.
Atributo	multiple
Definição	Permite que mais de uma opção seja selecionada pelo usuário.

d) formulário <form>...</form>

Os formulários são de grande utilidade na Web, pois permitem a interatividade entre o usuário e o Servidor Web.

A finalidade do formulário é coletar dados fornecidos pelo usuário, podendo com isso ler e gravar dados em um Banco de Dados.

O esquema de ações disparadas por um formulário HTML é visto na Figura 9, apresentada a seguir:

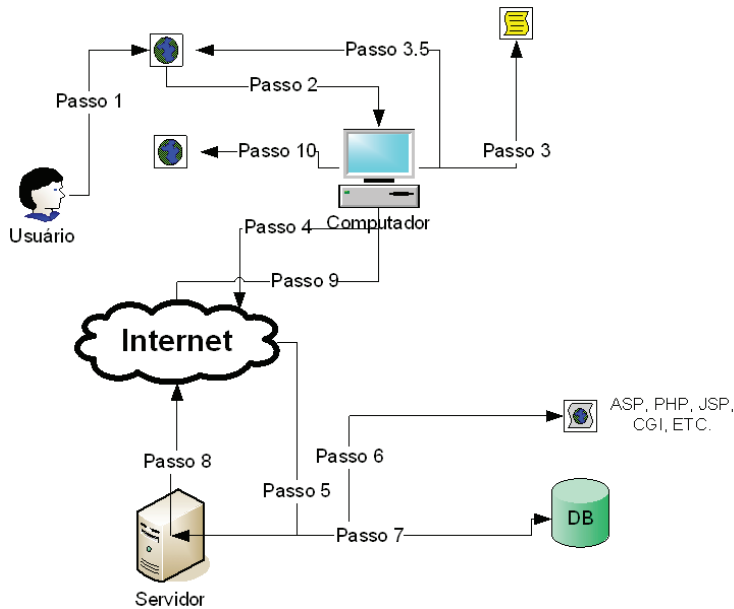


Figura 9 - Formulário

Estas ações podem ser assim descritas:

Passo 1: O usuário preenche os dados do formulário e aciona o botão **submit** para enviar os dados ao servidor.

Passo 2: O navegador pega os dados do formulário.

Passo 3: Caso tiver scripts associados ao formulário, estes são executados.

Passo 3.5: Dependendo do script, o navegador poderá alertar o usuário de algum erro de preenchimento e voltamos ao PASSO 1.

Passo 4: O navegador envia os dados para o servidor de páginas (**servidor WWW**) via Internet, através de uma requisição HTTP, solicitando como resposta uma página.

Passo 5: Os dados do formulário chegam ao servidor e são manipulados pelo servidor de páginas que os repassa à página mencionada no campo *action* do formulário.

Passo 6: Caso a página que recebeu os dados do formulário contenha algum script, o servidor o executa.

Passo 7: Caso o script precise acessar o Banco de Dados, então o acesso é realizado.

Passo 8: O servidor de páginas envia para o computador a página solicitada.

Passo 9: O computador recebe a página solicitada pelo formulário.

Passo 10: O navegador exibe os dados da página recebida.

2.3.5.1 Alguns aspectos importantes sobre um formulário

1. Delimita-se uma área de interação através das tags

`<form>` e `</form>`;

2. Os atributos `action` (**pra quem**) e `method` (**como**) devem ser definidos;

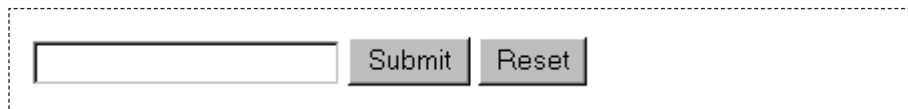
3. Dentro do form, são colocados os elementos de interação da HTML, formando assim os campos do formulário;

4. O formulário deve conter um botão que ative a ação de **enviar (“postar”) o formulário**. Ou seja, enviar os **dados preenchidos pelo usuário** para que o script (cgi, asp, php, jsp, etc.) possa processar esses dados e retornar um resultado, conforme programação pré-estabelecida no script.

Primeiro Exemplo:

```
<form method="POST" action="processa.php">
<p>
  <input type="text" name="T1" size="20">
  <input type="submit" value="Submit" name="B1">
  <input type="reset" value="Reset" name="B2">
</p>
</form>
```

Resultado:



The image shows a visual representation of the HTML form code. It consists of a rectangular box with a dashed border. Inside the box, there is a text input field on the left, followed by two buttons: "Submit" and "Reset".

O atributo `action` indica a **localização (URL)** do script que irá receber e interpretar os dados enviados pelo formulário.

No exemplo é chamado o script `processa.php` que se encontra no mesmo diretório da página HTML que contém o formulário.

Outra forma de indicar esse script seria colocar o endereço completo do script: `http://www.meusite.com.br/sistema1/processa.php`.

O atributo *method* indica o formato no qual os dados serão enviados. Pode assumir os valores “GET” (indica como os dados serão passados pelo script) ou “POST” (envia os dados para entrada padrão do sistema operacional), ambos são métodos do protocolo HTTP. O padrão, se omitido, é “GET”.

Caso seja usado o *method* “GET” ou omitido este atributo, o navegador junta os valores de todos os controles no formulário em uma cadeia de consulta e a anexa ao URL da página sendo solicitada.

Por meio deste método, os dados constantes no formulário são primeiramente transmitidos ao *software* servidor e este, por sua vez, armazena os dados temporariamente numa variável de contexto denominada **QUERY_STRING**. Quando um formulário HTML utiliza o método GET, o fluxo de dados é separado do endereço URL que chama o script através de um ponto de interrogação (?).

Esta forma de endereçamento e separação pode ser observada no campo de endereços do navegador, logo após o formulário ter sido enviado.

Por exemplo:

```
http://www.meusite.com/meuscript.php?nome=Maria&id=123
```

O método POST faz com que os dados do formulário sejam diretamente transmitidos ao endereço que constar na diretiva *action*.

Dessa forma, o script executado precisa extrair os dados através da entrada padrão (**standard input**) para poder obter os dados transmitidos pelo formulário. Sendo assim, todos os formulários deveriam usar o método POST.

Existem limites impostos pelo navegador e servidor quanto ao comprimento da cadeia do URL, pois anexar uma cadeia de consulta longa pode provocar um *overflow* e alguns dos valores podem ficar truncados. Também, a cadeia de consulta aparece na barra de endereços do navegador e em todos os favoritos e *links* gravados.

O cadeiamento de valores não apenas é feito, mas também expõe valores que não podem aparecer na solicitação HTTP quando passa para Web.

2.3.5.2 Exemplos de formulários

one-line text box

```
<form method="POST" action="_URL_">
  <input type="text" name="T1" size="20">
  <input type="submit" value="Submit" name="B1">
  <input type="reset" value="Reset" name="B2">
</form>
```



A screenshot of a web form. It features a single-line text input field on the left, followed by two buttons: "Submit" and "Reset".

scrolling text box

```
<form method="POST" action="_URL_">
  <textarea rows="2" name="S1" cols="20"></textarea>
  <input type="submit" value="Submit" name="B1">
  <input type="reset" value="Reset" name="B2">
</form>
```



A screenshot of a web form. It features a multi-line text area (scrolling text box) on the left, followed by two buttons: "Submit" and "Reset".

checkbox

```
<form method="POST" action="_URL_">
  <input type="checkbox" name="C1" value="ON">
  Tópico 1
  <input type="submit" value="Submit" name="B1">
  <input type="reset" value="Reset" name="B2">
</form>
```



A screenshot of a web form. It features a checkbox followed by the text "Tópico 1", and then two buttons: "Submit" and "Reset".

checked

```
<form method="POST" action="--WEBBOT-SELF--">
  <input type="checkbox" name="C1" value="ON" checked>
  <input type="submit" value="Submit" name="B1">
  <input type="reset" value="Reset" name="B2">
</form>
```

radio button

```
<form method="POST" action="_URL_">
  <input type="radio" value="V1" checked name="R1">
  Tópico 1
  <input type="submit" value="Submit" name="B1">
  <input type="reset" value="Reset" name="B2">
</form>
```

Tópico 1

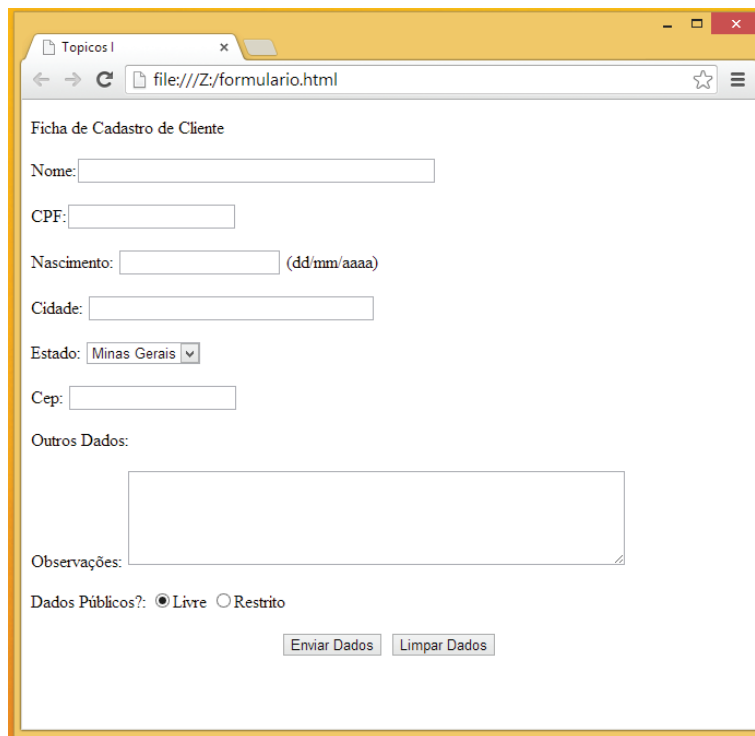
drop-down menu

```
<form method="POST" action="_URL_">
  <select name="D1" size="1">
    <option selected
value="http://www.xyz.com.br/topico1"> Tópico 1 </option>
    <option
value="http://www.xyz.com.br/topico1"> Tópico 2 </option>
  </select>
  <input type="submit" value="Submit" name="B1">
  <input type="reset" value="Reset" name="B2">
</form>
```

O exemplo apresentado na Figura 10 ilustra um formulário básico para cadastro de usuários visualizado no navegador Chrome.

Algumas considerações importantes:

- O atributo *action* faz referência a um arquivo chamado *verifica.php*, que deve estar no mesmo diretório do formulário.
- A validação dos dados do formulário não serão comentadas nesse momento, mas podem ser realizadas pela linguagem de script denominada JavaScript.



The image shows a web browser window with a single tab titled "Topicos 1". The address bar displays "file:///Z:/formulario.html". The page content is a registration form titled "Ficha de Cadastro de Cliente". The form includes the following fields and controls:

- Nome:
- CPF:
- Nascimento: (dd/mm/aaaa)
- Cidade:
- Estado: (dropdown menu)
- Cep:
- Outros Dados:
- Observações:
- Dados Públicos?: Livre Restrito
- Buttons: "Enviar Dados" and "Limpar Dados"

Figura 10 - Visualização do formulário no navegador

Código Fonte	Observações
<pre><html> <head> <title>Tópicos I</title> </head> <body> <form name="form1" method="post" action="verifica.php"> <p>Ficha de Cadastro de Cliente</p></pre>	
<pre><p>Nome:<input type="text" name="nome" size="49"> </p> <p>CPF:<input type="text" name="cpf" size="20"></p> <p>Nascimento: <input type="text" name="nascimento" size="19"> (dd/mm/aaaa)</p> <p>Cidade: <input type="text" name="cidade" size="38"> </p></pre>	Campos texto (nome , cpf , nascimento e cidade) com nome e tamanho definidos.
<pre><p>Estado: <select size="1" name="estado"> <option value="AC">Acre</option> <option value="AL">Alagoas</option> <option selected value="MG">Minas Gerais</ option> <option value="TO">Tocantins</option> </select> </p></pre>	<p>Apresenta uma lista de estados para que o usuário selecione um dentre os estados listados.</p> <p>O estado escolhido como default é Minas Gerais.</p> <p>Ao selecionar um estado, automaticamente o valor (value) da opção selecionada é atribuído à variável estado (name)</p>
<pre><p>Cep: <input type="text" name="cep" size="20"></p> <p>Outros Dados:</p></pre>	
<pre><p>Observações: <textarea rows="5" name="descricao" cols="54"></textarea> </p></pre>	Área de texto, chamada descrição , com 5 linhas e 54 colunas de visualização, mas com o auxílio da barra de rolagem o usuário pode digitar um texto maior.
<pre><p>Dados Públicos?: <input type="radio" name="acesso" value="livre" checked >Livre <input type="radio" name="acesso" value="restrito">Restrito </p></pre>	Inicialmente a opção "Livre" aparece selecionada (checked). Ao selecionar a opção "Restrito" a variável acesso recebe o valor restrito .
<pre><p align="center"> <input id="enviar" type="submit" value="Enviar Dados" name="enviar"> <input id="limpa" type="reset" value="Limpar Dados" name="limpa"> </p></pre>	<p>Definição dos botões para:</p> <ul style="list-style-type: none"> - Enviar o formulário (submit) para o endereço especificado no action. - Limpar os campos do formulário (reset)
<pre></form> </body> </html></pre>	

DICAS:

1. Cada campo do formulário deve ter um nome único. Portanto, utilize nomes que lembrem o que o campo significa, como: cep, cpf, rua, etc.
2. Campos do tipo "hidden" definem campos não visíveis que passam parâmetros para o script a ser executado. `<INPUT TYPE="hidden" NAME="nome-do-campo" Value="valor">`

2.3.6 Caracteres Especiais

A linguagem HTML faz uso do conjunto básico de caracteres da tabela ASCII, ou seja, os caracteres sem acentuação, números e símbolos padrão da escrita inglesa.

Outros símbolos a serem representados devem seguir uma regra de sintaxe específica, a saber: um & inicial, um número ou cadeia de caracteres correspondente ao caractere desejado e um ; final. Exemplo: ` ` que representa o caractere de espaço: ` `.

Alguns outros exemplos:

Entidade	Caractere	Entidade	Caractere
<code>&aacute;</code>	á	<code>&Aacute;</code>	Á
<code>&acirc;</code>	â	<code>&Acirc;</code>	Â
<code>&agrave;</code>	à	<code>&Agrave;</code>	À
<code>&atilde;</code>	ã	<code>&Atilde;</code>	Ã
<code>&ccedil;</code>	ç	<code>&Ccedil;</code>	Ç
<code>&eacute;</code>	é	<code>&Eacute;</code>	É
<code>&ecirc;</code>	ê	<code>&Ecirc;</code>	Ê
<code>&iacute;</code>	í	<code>&Iacute;</code>	Í
<code>&oacute;</code>	ó	<code>&Oacute;</code>	Ó
<code>&ocirc;</code>	ô	<code>&Ocirc;</code>	Ô
<code>&otilde;</code>	õ	<code>&Otilde;</code>	Õ
<code>&uacute;</code>	ú	<code>&Uacute;</code>	Ú
<code>&uuml;</code>	ü	<code>&Uuml;</code>	Ü

Caso digitar os caracteres acentuados normalmente (sem o uso do recurso de representação) provavelmente os caracteres aparecerão normalmente no seu navegador, mas quando a página for acessada por computadores configurados em outras línguas os caracteres aparecerão modificados, como apresentados na Figura 11 e na Figura 12.

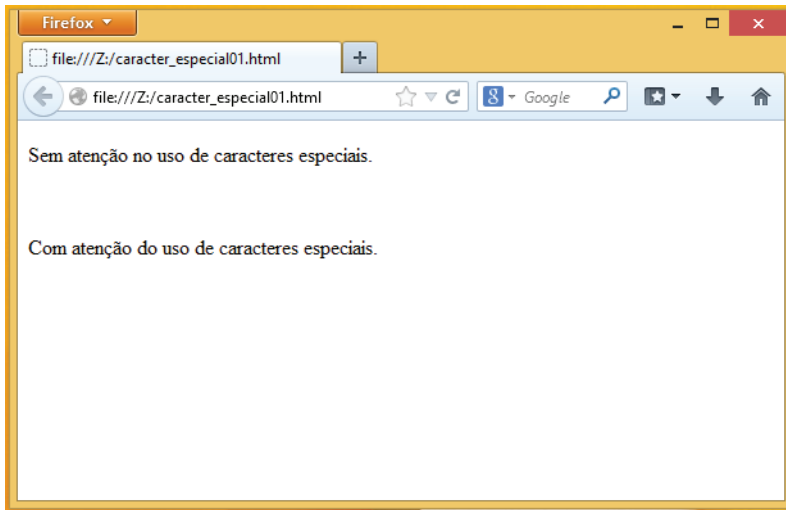


Figura 11 – charset UTF-8

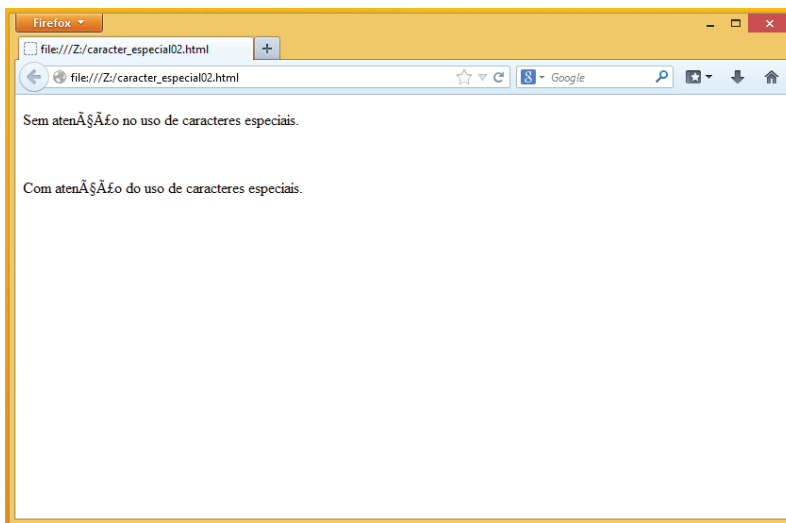


Figura 12 – Outra especificação de conjunto de caracteres

Ao utilizar editores que auxiliam na codificação do HTML, os editores já fazem a conversão entre o que você digita na área de edição e o código gerado, favorecendo a usabilidade. Entretanto, outro recurso normalmente utilizado pelos editores HTML é colocar a informação no próprio arquivo HTML de qual especificação do conjunto de caracteres que deve ser utilizado.

Veja o exemplo a seguir ilustrando o uso do conjunto de caracteres do padrão ISO 8859-1 e um com o padrão charset UTF-8.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>Título da página</title>
  </head>
<body>
  Teste de conteúdo
</body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Título da página</title>
  </head>
<body>
  Teste de conteúdo
</body>
</html>
```

2.4 Folhas de Estilo

As **Folhas de Estilo** ou **Cascading Style Sheets** ou apenas **CSS** são os nomes que podem ser usado para designar um conjunto de regras para formatar documentos HTML.

Com o **CSS** é possível definir o posicionamento de elementos, formatação de texto, posicionamento e tamanho de imagens, criação de menus e diagramação do *layout* de um documento HTML. Sendo assim, um mesmo **CSS** pode ser invocado várias vezes em diversas páginas de um mesmo website, garantindo um *layout* consistente e de fácil manutenção.

O padrão proposto pela W3C, o CSS, visa controlar a formatação visual de uma página Web. O controle visual permite o desenvolvimento do código HTML visando a separação em camadas garantido flexibilidade para a equipe de desenvolvimento.

Assim, **o HTML é o responsável** pela marcação e correta compreensão do dado pelos mecanismo de busca, **o CSS é o responsável** por controlar como o

layout de uma página pode ser apresentado incluindo cores e diagramação, e a **linguagem de script JavaScript** permite a interação de elementos do HTML e do CSS.

O **CSS** oferece várias possibilidades que antes só era possível com a utilização de *gifs* e *jpgs*. Basicamente, o conjunto de regras permite ao desenvolvedor um controle maior sobre os atributos tipográficos, tamanho e cor das fontes, espaçamento entre linhas, espaçamento entre os caracteres, margem do texto, utilização de *layers* que permite a sobreposição de texto sobre texto ou texto sobre figuras, entre outros recursos.

A inserção de estilo em um documento HTML pode ser feito da seguinte forma:

- **Local: Somente o elemento h1** que tem o atributo *style* definido, tem o tamanho da fonte modificado para 12px. Veja o CÓDIGO 2.4 apresentado a seguir.

CÓDIGO 2.4 – CSS aplicado localmente

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Estilo inline</title>
  </head>
  <body>
    <h1 style="font-size:12px">Parágrafo</h1>
  </body>
</html>
```

- **Geral: Todos os elementos h1** do documento HTML tem o tamanho da fonte modificado para 30 pixels. Veja o CÓDIGO 2.5 apresentado a seguir.

CÓDIGO 2.5 – CSS aplicado de forma geral

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Estilo embutido</title>
    <style>
      h1 { font-size: 30px;}
    </style>
  </head>
  <body>...</body>
</html>
```

- **Global:** O conjunto de regras de estilo são aplicados em todas as páginas que o arquivo “estilo01.css” é invocado. Veja o CÓDIGO 2.6 apresentado a seguir.

CÓDIGO 2.6 – CSS aplicado de forma global

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Estilo externo</title>
    <link rel="stylesheet" href="estilo01.css" type="text/
css"/>
  </head>
  <body>
    <h1>Parágrafo</h1>
  </body>
</html>
```

A ordem adotada para a aplicação do estilo é a seguinte:

- Primeiro se aplica o estilo local;
- Caso não tenha o local aplica-se o geral;
- E por último, caso não se tenha os outros dois estilos, aplica-se o estilo global.

Outra forma de importar o estilo é através do uso da tag @import. Entretanto, a tag @import foi idealizada para permitir a importação de estilos dentro de outro estilo, como recurso de complementação de um conjunto inicial de regras. Veja o exemplo do CÓDIGO 2.7:

CÓDIGO 2.7 – Uso da tag @import

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Estilo import</title>
    <style>
      <!--
        @import(estilo01.css);
      -->
    </style>
  </head>
  <body>
    <h1>Parágrafo</h1>
  </body>
</html>
```

A tag @import também pode ser usada como apresentada na Figura 13.

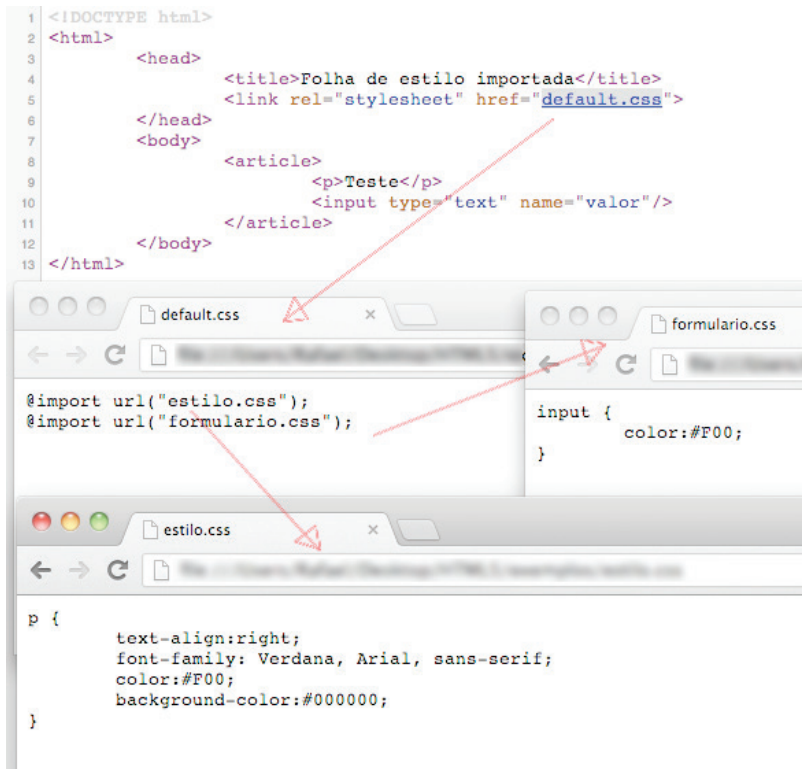


Figura 13 - Tag @import

O **CSS** permite um controle absoluto da aparência da página, o que não se tem com o código HTML. Pode-se colocar uma imagem em qualquer lugar da página, usando técnicas de posicionamento absoluto ou relativo. Além disso, pode determinar uma imagem ou cor para o fundo de uma página Web.

2.4.1 Declaração de regras de estilo

As Folhas de estilo permitem que sejam atribuídas propriedades de estilo através de regras aos elementos HTML. Por exemplo, considere a seguinte regra: “todo **elemento parágrafo** pertencente a uma **classe** denominada **editorial** será **azul**, terá o **tamanho da fonte definida em 12 pt**, **espaçamento duplo**, **alinhamento à direita** e usará a **família de fontes** Arial, ou, se esta não existir, Helvetica, ou então a fonte sem serifa default do sistema”.

A regra apresentada pode ser escrita conforme o CÓDIGO 2.8 apresentada a seguir:

CÓDIGO 2.8 – Regra em CSS

```
p.editorial {
  color:#0000ff;
  font-size:12pt;
  line-height:24pt;
  text-align:right;
  font-family:arial, helvetica, sans-serif;
}
```

Caso a regra for aplicada a uma página que possua elementos parágrafos com o atributo **class** e **valor** igual a **editorial**, os parágrafos serão formatados de acordo com as propriedades especificadas.

A estrutura das regras para criação de estilo é bastante simples. Consiste de uma lista de regras em que cada regra possui um bloco, entre chaves ({ e }), de uma ou mais declarações aplicáveis a um ou mais seletores.

Um seletor é algo no qual pode-se aplicar um estilo. Pode ser um elemento HTML, um identificador de algum elemento HTML ou uma classe que identifique um grupo de elementos.

Uma folha de estilos consiste de uma ou mais linhas de regras, da seguinte forma: `seletores { declarações }`.

As regras podem estar dentro de um arquivo de texto (ISO Latin1 ou ASCII 8-bit) com extensão “.css” ou embutidas em um arquivo HTML, como apresentado anteriormente.

Para definir a propriedade de uma regra usa-se os dois-pontos (:) e não igual (=). No mesmo conjunto de regra pode haver mais de uma declaração de estilo para um seletor, definindo um bloco de regra. Cada linha acrescenta ou sobrepõe declarações feitas em linhas anteriores, veja o exemplo a seguir:

```
h1 { font-size: 24pt }
h1 { color: blue }
h1 { font-size: 18pt }
```

ou

```
h1 {font-size: 24pt; color: blue; font-size: 18pt }
```

No trecho acima, o texto marcado com `<h1>` será azul e terá tamanho de 18pt, porque a regra `h1 { font-size: 18pt }`, ocorreu depois da regra `h1 { font-size: 24pt }`.

2.4.2 ID vs Classe vs Elemento

As regras a serem criadas podem ser agrupadas em identificadores, classes ou elementos.

Quando agrupados em identificadores o nome da regra é antecedido por uma cerquilha #.

Logo, no agrupamento por classes a regra é antecedido por um ponto.

No agrupamento de regras por elemento é simplesmente informado o nome do elemento e na sequência as propriedades e valores que devem ser aplicadas a este elemento.

Para definir se deve ser criado uma regra por elemento, identificador ou classe deve ser analisado como a folha de estilo está sendo construída. Isso porque quando define uma regra por elemento isso significa que todos os elementos tem o estilo modificado.

Assim, quando cria-se uma regra por classe significa que todos os elemento que tem o atributo *class* com o valor igual ao nome da classe passa a ter o estilo modificado.

Por fim, ao criar uma regra considerando o identificador, ou seja, um nome único atribuído a um elemento mapeado pelo atributo *id* significa que somente o elemento com aquele nome tem o estilo modificado.

Veja a seguir alguns exemplos de criação de regras de folha de estilo apresentado no CÓDIGO 2.9, CÓDIGO 2.10 e CÓDIGO 2.11.

CÓDIGO 2.9 - Identificador

```
<style>
#logotype {
    margin-top:2px;
    float:left;
}
</style>

<div id="logotype">

</div>
```

CÓDIGO 2.10 - Elemento

```
<style>
header {
    width:757px;
    margin:0 auto;
```

```

        height:97px;
    }
</style>

<header>
    <span id="logotipo"></span>
    <nav>
        <ul id="menu">
            <li><a href="#">Gerenciar</a></li>
            <li><a href="#">Fale Conosco</a></li>
        </ul>
    </nav>
</header>

```

CÓDIGO 2.11 - Classe

```

<style>
    .cor { color: #D90B17; }
</style>
<h1 class="cor">Vermelho</h1>

```

Existem várias propriedades que podem ser usadas em uma regra. A finalidade deste tópico nesta unidade não é abranger todas as propriedades.

Assim, conforme forem sendo montadas as regras as propriedades serão explicadas.

Nos exemplos apresentados não se preocupe com as regras, apenas entenda os tipos de agrupamento classe, elemento e identificador.

Na seção prática deste tópico, as propriedades das regras serão melhores explicadas.

2.4.3 Fontes

A seguir são apresentadas algumas propriedades para manipular fontes.

font-family: Propriedade para indicar a fonte a ser utilizada.

```

p {
    font-family:arial, helvetica, sans-serif;
}

```

No exemplo acima, tudo o que estiver entre as tag **<p>** e **</p>** no seu documento HTML será visualizado em Arial. A razão de definir mais de uma fonte deve-se ao fato de que nem todos os computadores possuem as mesmas fontes instaladas.

No caso, se o computador não possuir a fonte Arial, automaticamente pasará para a segunda opção e mostrará o texto em Helvetica.

font-size: Propriedade para definir o tamanho da fonte. Há três formas básicas para definição de tamanho:

1. pt (pontos), px (pixels), in (inches), cm (centímetros), mm (milímetros), pc (picas) e em.

```
p { font-size: 20pt; }
```

2. xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger.

```
p { font-size: medium; }
```

3. Percentagem: O tamanho das fontes também pode ser determinado por um valor em percentual

```
p { font-size: 50%; }
```

font-weight: Propriedade para controlar o negrito de um elemento.

```
h1 { font-weight: bold; }
```

Observação:

- O **font-weight** pode ser configurado com o valor “normal” para desativar todos os **bolds** de uma página.
- O valor da propriedade pode ser determinado em números de 100 a 900, sendo 900 o valor máximo de negrito disponível.

font-style: Propriedade para determinar o valor itálico de um elemento.

```
h1 { font-style: italic; }
```

No exemplo acima o navegador busca uma versão itálica da fonte para apresentar o texto compreendido entre as tags `<h1>` e `</h1>`.

2.4.4 Cores

Para mudar a cor de um texto usa-se a **propriedade color**. As cores podem ser aplicadas em qualquer elemento de uma página HTML e podem ser definidas por **nomes**, **números hexadecimais** e **valores RGB**, como apresentado no CÓDIGO 2.12 a seguir:

CÓDIGO 2.12 – Trabalhando com cores em CSS

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Estilo</title>
    <style>
      p.azul {
        color: blue;
      }

      p.amarelo {
        color: #FFD800;
      }

      p.vermelho {
        color: rgb(255,0,0);
      }
    </style>
  </head>
  <body>
    <p class="azul">Parágrafo</p>
    <p class="amarelo">Parágrafo</p>
    <p class="vermelho">Parágrafo</p>
  </body>
</html>
```

2.4.5 Posicionamento

A folha de estilo permite posicionar os elementos dentro de um documento HTML de uma forma mais flexível que o próprio HTML. Assim, a folha de estilo ajuda a resolver esses problemas e habilita novos recursos para criação de *layout* melhores.

As propriedades **position**, **left** e **top** são utilizados para posicionar os elementos. A propriedade **left** determina a distância entre o elemento e a margem esquerda da página. A propriedade **top** determina a distância até a parte superior da mesma.

Os valores das propriedades podem ser informados em porcentagem ou unidades de medida que podem ser as mesmas mencionadas anteriormente.

A propriedade **position** determina se o posicionamento dos elementos será **absoluto** ou **relativo**. **Posicionamento absoluto** significa que cada elemento tem a sua posição determinada individualmente, através de coordenadas.

Já no **posicionamento relativo** a posição de cada elemento é determinada em relação aos outros.

Qualquer elemento pode ter sua posição determinada por uma folha de estilo: imagens, vídeos, parágrafos ou uma única letra.

Além dessas propriedades, existe a propriedade **width** para determinar a largura que pode ser especificada em porcentagem ou unidade de medida. A propriedade de largura só pode ser aplicada a elementos com posicionamento absoluto em uma página.

Por fim, existe a propriedade **z-index** que possibilita sobrepor vários elementos definindo em que ordem os elementos serão visualizados, através do posicionamento absoluto como apresentado no CÓDIGO 2.13.

CÓDIGO 2.13 – Posicionamento com CSS

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Posicionamento</title>
    <style>
      h1 { float: right; }
      h2 { position: absolute;
          top: 120px;
          left: 130px;
          color: #FF0200; }
    </style>
  </head>
  <body>
    <h1>Cabeçalho 01</h1>
    <h2>Cabeçalho 02</h2>
    <h2>Cabeçalho 02</h2>
  </body>
</html>
```

2.4.6 Prática

Para praticar a criação de folhas de estilo é apresentado o conjunto de regras de estilo do cabeçalho da página Web a ser usado no sistema de gerenciamento de notícias. Assim, use o NetBeans para criar a página que vai ser apresentada a seguir.

O sistema de gerenciamento de notícias é composto de um cadastro completo (**criação, alteração, exclusão e visualização**) que servirá como base para o desenvolvimento do sistema proposto durante o livro ao aluno.

As funcionalidades de inclusão, exclusão, alteração e visualização serão abordados nas próximas unidades.

Neste momento, o foco é apenas a página inicial da aplicação que permite selecionar as opções de entrar em contato e gerenciar as notícias.

Assim, na Figura 14, Figura 15 e Figura 16 são apresentados como a página é visualizada no Firefox, Chrome e IE. Na sequência, o código HTML e a folha de estilo são apresentados.

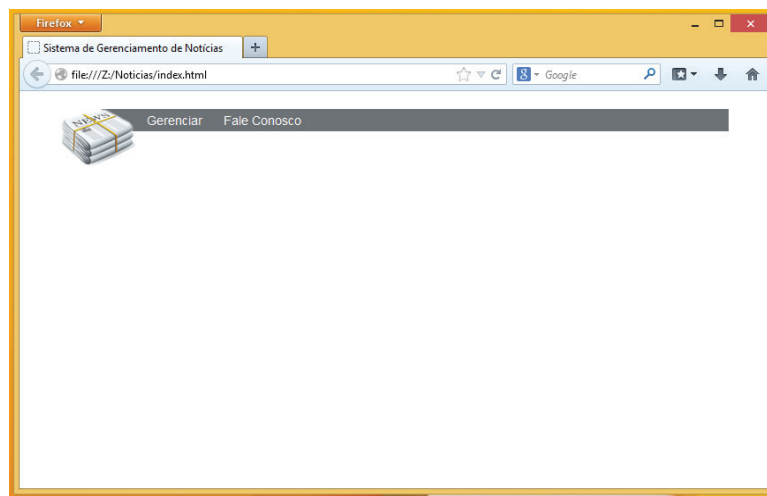


Figura 14 (Firefox) Visualização da página inicial do sistema de gerenciamento de notícias.

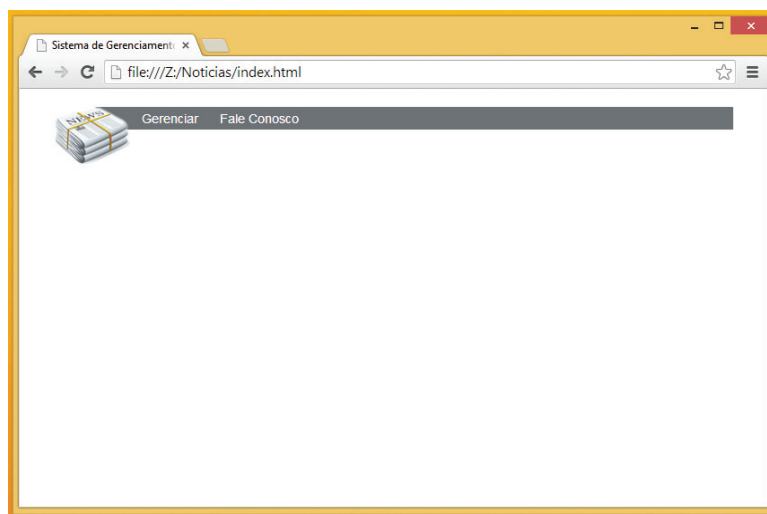


Figura 15 (Chrome) Visualização da página inicial do sistema de gerenciamento de notícias.

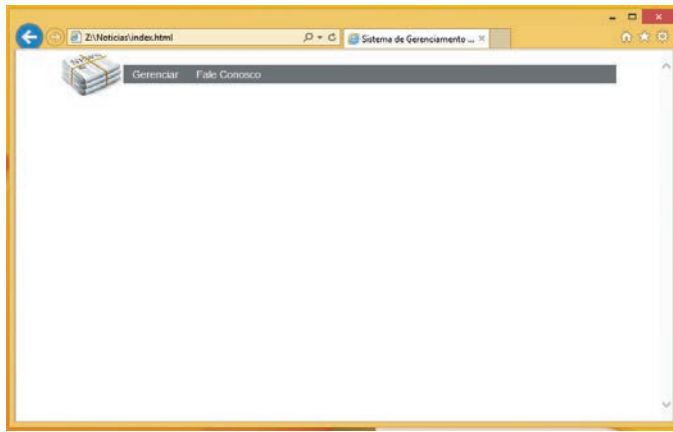


Figura 16 (IE11) - Visualização da página inicial do sistema de gerenciamento de notícias.

Uma observação a ser feita é que para o código funcionar corretamente no IE foi inserido uma biblioteca JavaScript oferecida pela Google para corrigir problemas no IE em relação ao uso do HTML5. O nome da biblioteca é o HTML5SHIV. Assim, apenas insira o seguinte código no cabeçalho como apresentado no CÓDIGO 2.15.

CÓDIGO 2.15 – Inserção da biblioteca HTML5SHIV

```
<!--Garante o suporte correto do HTML5 para o Internet Explorer-->
<!--[if lt IE 9]>
<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
```

Após inserir o código acima entre as tags head da sua página Web, o IE entende a existência das novas tags usadas tais como a tag **<header>**.

Para criar o cabeçalho apresentado nas Figuras 14, 15 e 16 foi desenvolvido o CÓDIGO 2.16 em HTML5 e CSS apresentado a seguir.

CÓDIGO 2.16 – Menu do Sistema de Gerenciamento de Notícias

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Sistema de Gerenciamento de Notícias</title>
    <!--Garante o suporte correto do HTML5 para o Internet Explorer-->
    <!--[if lt IE 9]>
```

```

<script src="http://html5shiv.googlecode.com/svn/
trunk/html5.js"></script>
<![endif]-->

<!-- Criação do estilo da página usando CSS
incorporado -->
<style>
/*
    Define a margem e o espaçamento com o valor igual a
    zero.

    A propriedade margin refere-se a margem e ao aplicar
    o valor igual a zero tanto no topo, embaixo, lado direito
    e esquerdo da imagem a margem é igual a zero.

A propriedade padding define um espaçamento do conteúdo
com as bordas ao retorno do elemento. Neste caso, quando
definido 0 significa que todos os elementos tem o padding
tanto em cima, abaixo, lado direito e lado esquerdo igual
a zero.
*/
* {
    margin:0;
    padding:0;
}

/*
    Define a fonte da página em 88% e fonte igual a Arial,
    Helvetica e Sans-serif
*/
body {
    font:88% Arial, Helvetica, sans-serif;
}

/*
    A propriedade text-decoration pertencente a tag "a"
    que define um link no documento HTML, quando atribuída com
    o valor none retira o sublinhado.
*/
a {
    text-decoration: none;
}

/*
    A propriedade list-style-type retira os bullets de
    uma lista. Entende-se por bullets o desenho ou símbolo que
    fica ao lado esquerdo de uma lista.
*/

```

```
ul,ol {
    list-style-type:none;
}
```

```
/*
```

A propriedade width define a largura do cabeçalho representado pela tag "header" do HTML5 em 757px de largura.

A propriedade margin define as margens desse elemento. Entende-se margem como sendo o espaçamento entre o topo, lado direito, lado esquerdo e abaixo de um elemento. Quando a margem está automática significa que o conteúdo fica centralizado na página.

```
*/
```

```
header {
    width: 757px;
    margin: auto;
}
```

```
/*
```

A propriedade background-color define a cor de fundo do elemento nav em cinza. Assim, as cores em folha de estilo (CSS) são definidas em Hexadecimais.

A propriedade height define a altura em 25px, dessa forma define uma espessura a barra criada em folha de estilo.

```
*/
```

```
nav {
    background-color: #6d7276;
    height: 25px;
}
```

```
/*
```

A propriedade float define o alinhamento do elemento que está com o ID #logotipo. Assim, quando o valor da propriedade é igual a left indica que o elemento é lançado à esquerda.

A propriedade margin-top define o espaçamento em relação ao topo do navegador. Quando definido com o valor negativo como o exemplo a seguir, mostra que a imagem sobe.

```
*/
```

```
#logotipo {
    float: left;
    margin-top: -15px;
}
```

```
/*
```

A propriedade margin-top define o valor de 20px, ou seja está à 20px em relação ao topo do navegador neste caso.

```
*/
nav {
    margin-top: 20px;
}

/*
    Uma lista é formada por uma lista de itens e cada item
    deve ser alinhado à esquerda. Quando a propriedade left é
    atribuída ao elemento li significa esse comportamento.
*/
#menu li {
    float:left;
}

/*
Quando encontrar um ID definido como #menu e que tenha a
tag a é aplicado as seguintes regras:
- Atribuir um padding de 5px margem top;
- Atribuir um padding de 5px margem bottom;
- Atribuir um padding de 12px margem right;
- Atribuir um padding de 12px margem left;
- Atribuir a cor em branco;
- Atribuir a aparência em bloco;
*/
#menu a {
    padding: 5px 12px;
    color: #FFFFFF;
    display:block;
}

/*
    Quando o usuário passa o mouse em cima do link aciona
    a propriedade hover. Assim, a cor da fonte deve ser preta
    (#000000), a cor de fundo branca (#FFFFFF) e deve ser
    apresentado como bloco (display:block). */
#menu a:hover {
    background-color:#FFFFFF;
    color:#000000;
    display:block;
}
</style>
</head>
<body>
<header>
<span id="logotipo"></span>
```

```
<nav>
  <ul id="menu">
    <li><a href="#">Gerenciar</a></li>
    <li><a href="#">Fale Conosco</a></li>
  </ul>
</nav>
</header>
</body>
</html>
```

2.5 JavaScript

A linguagem de **marcação HTML** deve ser usada especificamente para estruturar documentos. Já questões de **estilo** deve ficar a cargo da **folha de estilo (CSS)** como já exposto neste material.

Outro ponto é a inserção de interatividade em uma página HTML. Para isso, usa-se linguagens de programação.

Dentre as linguagens existentes, uma linguagem que pode ser usada para adicionar interatividade é a linguagem de script que roda no lado do cliente (**client side**) que visualiza a página, denominada JavaScript.

Criada por **Brendan Eich** e lançada juntamente com o navegador **Netscape Navigator 2.0** em 1996, permite o desenvolvedor adicionar interatividade a uma página Web.

A linguagem JavaScript depende exclusivamente do navegador do usuário que está visualizando a página. Sendo assim, as funcionalidades e os recursos são suportados pelas máquinas de interpretar scripts denominados **engines**. Por exemplo, o navegador Firefox tem o engine **SpiderMonkey** escrito na linguagem C/C++, o **Google Chrome** tem o engine **V8** escrito na linguagem C/C++ e existem vários outros engines.

A linguagem JavaScript tem sido extensivamente usada na área de desenvolvimento de páginas Web. Existem frameworks e bibliotecas para lidar com a criação de páginas Web usando JavaScript. A proposta deste tópico é apresentar a sintaxe da linguagem e como é possível acessar elementos html usando o JavaScript.

2.5.1 Funcionalidades

Ao usar a linguagem JavaScript o desenvolvedor pode **manipular e alterar o conteúdo** da página Web, **alterar características do navegador** e interagir

com os formulários criados em HTML, podendo acessar os campos e adicionar validações por exemplo.

2.5.2 Inserção de JavaScript no HTML

A linguagem JavaScript é incluída dentro de uma página Web. Para isso, deve ser usado um elemento denominado **script**. Tal elemento tem a função de incorporar um script ou invocar um arquivo de extensão “.js” para dentro do corpo da página Web.

O elemento **script** tem um atributo denominado **src**, este **atributo** tem a finalidade de **indicar o caminho que o script está localizado no servidor por meio de uma URL**.

No **HTML5**, o elemento `script` é codificado da seguinte forma:

```
<script src="arquivo.js"></script>
```

A forma de codificação apresentada permite chamar arquivos de **extensão “.js”** para dentro da página Web. Além dessa forma é possível incorporar o **script dentro do HTML**. Ou seja, entre as tags `<script>...</script>` é possível definir o código a ser implementada na página, como apresentado no **CÓDIGO 2.17**.

CÓDIGO 2.17 – Inserção do código JavaScript na página HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Incorporado</title>

    <script>
      alert('Olá Mundo!');
    </script>
  </head>
  <body>
  </body>
</html>
```

Uma outra forma também usada é a inserção do JavaScript de **forma inline**. Trata-se de uma forma de inserir o script diretamente na seção **body**, misturando com os elementos de estrutura do HTML, como apresentado no **CÓDIGO 2.18**.

CÓDIGO 2.18 – Inserção código javascript inline

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Inline</title>
  </head>
  <body>
    <button type="button" onclick="alert('Olá
Mundo!');">Olá</button>
  </body>
</html>
```

2.5.3 Código JavaScript

Adicionar um script de forma **inline**, **externa** ou **incorporado** é um primeiro passo para criação de scripts. Entretanto, como deve ser escrito o código usando a linguagem JavaScript ?

O primeiro ponto que deve ser considerado é que a linguagem JavaScript é sensível em relação a caixa de texto – *case sensitive*.

Em outras palavras, **nomes de variáveis** com letras **minúsculas** e **maiúsculas** são diferentes, assim como escrever as palavras reservadas para criar o código também faz diferença.

Exemplo:

Total é diferente de **total**
document.Write() (gera erro) é diferente de **document.write()**

Faz parte da sintaxe da linguagem a possibilidade de criar comentários no código, garantindo esclarecimento de pontos críticos da programação ao desenvolvedor.

Os comentários podem ser colocados em uma única linha e múltiplas linhas, como apresentado no CÓDIGO 2.19.

CÓDIGO 2.19 – Linhas de comentário

```
<!DOCTYPE html>
<html>
  <head>
    <title>Comentários</title>
    <meta charset="utf-8"/>
    <script>
      // Comentário de uma linha
      <!-- Comentário em linha única
      /* Comentário para múltiplas linhas*/
      alert("Inserir comentário!");
    </script>
  </head>
  <body>
  </body>
</html>
```

Para inserir algum tipo de regra ao script é necessário recorrer as declarações e estruturas de controle, pois script nada mais são do que sequências de instruções com base em uma sintaxe rígida.

O código em JavaScript pode ser escrito tudo em uma mesma linha, ou em linhas separadas. Sendo que quando o código é escrito em uma mesma linha é necessário usar o **ponto-e-vírgula** para separar o código.

No entanto, o uso de ponto-e-vírgula é facultativo quando o código está escrito em linhas separadas.

Uma outra característica é o espaço em branco que é ignorado quando inserido entre a declaração de variáveis, funções e números, observe o CÓDIGO 2.20.

CÓDIGO 2.20 – Sintaxe do código JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <title>Declaração</title>
    <script type="text/javascript">
      graduação = "BSI"; curso= "DSW2014";
      alert("Olá pessoal!");
    </script>
  </head>
  <body>...</body>
</html>
```

A criação de variáveis é importante em qualquer linguagem, principalmente porque é através das variáveis que os dados são manipulados e as regras podem ser incluídas nos scripts.

As variáveis e as constantes em JavaScript não possuem definição de tipos, pois o tipo fica implícito no valor que a variável ou a constante recebe.

Para criar uma variável nesta linguagem deve ser considerado algumas características relacionadas a criação do nome, ou seja, o nome da variável deve começar com uma letra ou caractere underline (`_`) ou cifrão (`$`). Na sequência pode seguir letra maiúscula ou minúscula, números, etc.

A **linguagem JavaScript** apresenta os seguintes tipos de dados: *inteiros*, *decimais*, *booleanos*, *strings*, *arrays* e *objeto*.

Um **tipo de dado inteiro** pode ser entendido com número fracionários ou inteiro, veja as possíveis variáveis recebendo número fracionários: `a=4.5`; `b=90`;

O **tipo de dado booleano (boolean)** recebem o valor `true` ou `false` que devem ser escritos em letras minúsculas. Por exemplo: `a=true`; `b=false`;

Agora o **tipo de dado string** trata-se de um ou mais caracteres envolvidos por aspas duplas ou simples. Por exemplo: `graduacao="BSI"`; `curso="DSW"`;

O **tipo de dado string** pode ser encarado como um objeto, podendo ser aplicado alguns métodos para manipular o texto armazenado. Alguns métodos contêm quantidade de caracteres `length`, colocar todas os caracteres em maiúsculo `toUpperCase()`, colocar todas os caracteres em minúsculo `toLowerCase()`, entre outros.

O **tipo de dado array** pode guardar qualquer tipo de dado, incluindo expressões, objetos e outros tipos de array. O array inicia-se com um índice sequencial começando do zero e tem a seguinte sintaxe:

```
carros = ["monza", "celta", "sonic"];
```

Para guardar mais de um tipo de dado, segue a seguinte sintaxe:

```
arrayMisturado = ["laranja", 2002, "casa", a+b, [1,2,3,4,5]];
```

A variável dependendo do escopo no código JavaScript pode ter o papel de **constante** ou **variável**. Geralmente para declarar uma variável pertencente ao **escopo local** usa-se o prefixo **var** antes da variável, como apresentado no CÓDIGO 2.21.

CÓDIGO 2.21 – Trabalhando com variáveis em JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <title>Escopo</title>
    <meta charset="utf-8"/>
    <script>
      numero = 100;

      funcaoUm = function() {
        var numero = -20;
        alert(numero);
      }

      //Imprimr o valor -20
      funcaoUm();

      funcaoDois = function() {
        alert(numero);
      }
      //Imprimir o valor 100
      funcaoDois();

      //Imprimir o valor 100
      alert(numero);
    </script>
  </head>
  <body>
  </body>
</html>
```

No código acima, o **valor da constante número 100** é apresentado dentro de uma mensagem quando a **função Dois** ou o **alert(numero)** são invocados. Isso acontece porque não ocorreu nenhuma alteração de valor. No entanto, ao invocar a chamada da **função Um** é apresentado o valor -20 porque dentro da função é criado uma variável local com o nome numero.

Para amarrar as variáveis e criar as regras precisa ser estudado como são codificados a estruturas condicionais e de loop na linguagem JavaScript.

As condições são criadas com uso do if que deve conter uma expressão seguido por chaves de abertura e fechamento, veja o exemplo apresentado no CÓDIGO 2.22.

CÓDIGO 2.22 – Condições em JavaScript

```
var media = 10;
if (media <= 10) {
  alert("Aluno reprovado!");
} else {
  alert("Aluno aprovado!");
}
```

Vale lembrar que `(media <= 10)` é uma expressão, e por se tratar de uma expressão pode ser combinado os diversos operadores que existem na linguagem que são apresentados no próximo tópico.

Em resumo, a estrutura condicional pode ser codificada da forma apresentada no CÓDIGO 2.23.

CÓDIGO 2.23 – Estrutura condicional

```
if ([expressao]) {
  /* Algo deve acontecer se a
   expressão resultar
   em uma verdade */
} else if ([expressao]) {
  /* Algo deve acontecer se a
   expressão resultar
   em algo falso. */
} ....
```

Além do `if`, existe a estrutura `switch` na linguagem JavaScript. A estrutura `switch` funciona como um chaveamento em que uma determinada variável é usada para identificar quais das condições do `switch` deve ter o código executado, veja a estrutura da sintaxe apresentada no CÓDIGO 2.24.

CÓDIGO 2.24 – Estrutura condicional usando switch

```
switch ([expressão]) {
  case resultado :
    //algo aqui;
    break;
  case resultado :
    //algo aqui;
    break;
  ...
  default:
    //comportamento padrão
}
```

As estruturas para controlar o loop estão mais relacionadas com o **for**, **while** e **do/while**. Em todas as estruturas é possível presenciar um condição para que o loop seja executado. Vale lembrar que o **do/while** tem uma característica diferente das demais, ou seja, a interação é executado pelo menos uma única vez.

Os CÓDIGOS 2.25, 2.26 e 2.27 apresentam a sintaxe dos três tipos de estrutura de loops:

CÓDIGO 2.25 - for

```
<!DOCTYPE html>
<html>
  <head>
    <title>FOR</title>
    <script type="text/javascript">
      var contar = "";
      for(var i=0; i<10; i++) {
        if ( i < 9) {
          contar += i+", ";
        } else {
          contar += i;
        }
      }
      document.write(contar);
    </script>
  </head>
  <body>
  </body>
</html>
```

CÓDIGO 2.26 - while

```
<!DOCTYPE html>
<html>
  <head>
    <title>WHILE</title>
    <script type="text/javascript">
      var contar = "";
      var i = 0;
      while(i < 10) {
        if (i < 9) {
          contar += i + ", ";
        } else {
          contar += i;
        }
        i++;
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

```
        document.write(contar);
    </script>
</head>
<body>
</body>
</html>
```

CÓDIGO 2.27 - do/while

```
<!DOCTYPE html>
<html>
  <head>
    <title>DO/WHILE</title>
    <script type="text/javascript">
      var contar = "";
      var i = 0;
      do {
        if ( i < 9) {
          contar += i+", ";
        } else {
          contar += i;
        }
        i++;
      } while(i < 10);
      document.write(contar);
    </script>
  </head>
  <body>
  </body>
</html>
```

Após apresentado a sintaxe, a forma como criar comentários, a forma como o código deve ser escrito, a criação de variáveis, os tipos de dados, o comportamento do escopo da variável em relação a declaração no código, a criação de estruturas condicionais e de loops é necessário conhecer os operadores que vão auxiliar na construção da lógica de programação do script.

2.5.4 Operadores

Ao estudar uma linguagem de programação é fato que operadores mais comuns tais como +, -, *, /, <, >, <=, >= estão presentes em grande parte das linguagens.

Em JavaScript é válido destacar que além da função de incremento que o operador + tem, este mesmo serve para concatenar strings. Exemplo: a = "4" + "5";

Além dos operadores mais comuns, citado anteriormente, o JavaScript apresenta os seguintes operadores:

Operador	Finalidade
==	Igualdade: compara se os símbolos são iguais – retorna true se os dados são iguais e false , caso contrário. Ao comparação é feita sem levar em conta o tipo de dado.
!=	Não igualdade: executa de forma contrário a função de igualdade.
&&	E (lógico): operador direito e esquerdo devem ser iguais para retornar true , caso contrário retorna false .
	OU (lógico): retorna true se um dos operandos for verdadeiro.
[]	Indexação de arrays.
()	Função.
?:	Operador ternário.
=	Atribuição de valor.
+=, - =, *=, /=	Atribuir valor executando alguma operação.
++, --	Pré ou Pós / Incremento (++) ou Pré ou Pós decremento (--) .
===	Identidade: compara os símbolos considerando o tipo de dado. Retorna true se forem iguais, e false caso contrário.
!==	Não identidade: executa de forma contrário a função de identidade.
!	Not: colocado antes de um operador nega a resposta.

Os operadores auxiliam o desenvolvedor na construção das regras que são aplicadas na linguagem JavaScript. Assim, o código JavaScript é composto por variáveis regidas por operadores que são controladas por estruturas condicionais ou loops.

O código por sua vez pode ser organizado dentro de funções e essas funções são criadas dentro dos arquivos de extensão “.js”, ou então, incorporado dentro da página Web.

2.5.5 Função

No desenvolvimento de código é normal existir trechos que se repetem. Os trechos por sua vez podem fazer parte de uma função.

A função trata-se de um bloco de código (corpo da função) que pode realizar cálculos complexos à manipulação de dados do script.

O uso da função ajuda a organizar o código de um script garantindo a manutenabilidade e legibilidade do código. Geralmente a função é criada com um nome que indica a ação a ser executada pela função.

Na linguagem JavaScript uma função pode ser criada de diferentes maneiras. A primeira dessas maneiras é a **função estática** que para ser executada precisa ser realizado a chamada direta da função, veja como codificar uma **função estática** analisando o CÓDIGO 2.28.

CÓDIGO 2.28 – Função estática

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Função estática</title>
    <script>
      function ola() {
        alert('Bem-vindo ao curso DSW do
BSI!');
      };
    </script>
  </head>
  <body>
    <button type="button" onclick="ola()">Função
estática</button>
  </body>
</html>
```

Outra forma de criar função é denominada de **função anônima ou dinâmica**, em que com o uso da declaração *new* seguida do construtor como apresentado no CÓDIGO 2.29.

CÓDIGO 2.29 – Função anônima ou dinâmica

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Função Anônimo</title>
    <script>
      var calculaAreaTriangulo = new
Function("b", "h", "return b*h/2;");
    </script>
  </head>
  <body>
    <button type="button" onclick="alert(calculaAre
aTriangulo(5,10));">Executar função</button>
  </body>
</html>
```

Além da **função anônima ou dinâmica** e a **função estática**, existe a função denominada **sintaxe literal**. A **função sintaxe literal** é conhecida como **funções expressões** e pode ser codificada como apresentado no CÓDIGO 2.30.

CÓDIGO 2.30 – Função sintaxe literal

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Função Sintaxe Literal</title>
    <script>
      var calculaAreaTriangulo = function(b,h) {
        return b*h/2;
      };
    </script>
  </head>
  <body>
    <button type="button" onclick="alert(calculaAreaTriangulo(5,10));">Executar função</button>
  </body>
</html>
```

As funções podem além de realizar cálculo e manipular dados, retornar valores. O retorno é feito com o uso da palavra **return** no corpo da função como nos exemplos das funções anônimas e de funções expressões. Uma função pode retornar um array, objeto e outros tipos de valores tais como texto ou número.

2.5.6 Caixa de alertas

A linguagem JavaScript permite criar **caixas de diálogos** do tipo **pop-up** permitindo coletar ou apresentar informações ao usuário.

Para apresentar uma caixa de diálogo com informação basta seguir a seguinte sintaxe do CÓDIGO 2.31.

CÓDIGO 2.31 – Caixa de diálogo

```
<!DOCTYPE html>
<html>
  <head>
    <title>Alerta</title>
    <script>
      alert("Teste");
    </script>
  </head>
  <body>
  </body>
</html>
```

Além de apresentar informações para o usuário é possível permitir alguma interatividade com as caixas de diálogo no JavaScript. Para isso, com o comando `confirm` o usuário pode selecionar se deseja aceitar ou cancelar a operação, o exemplo do CÓDIGO 2.32 apresenta o uso do comando `confirm`.

CÓDIGO 2.32 – Uso do comando `window.confirm`

```
<!DOCTYPE html>
<html>
  <head>
    <title>window.confirm</title>
    <meta charset="utf-8"/>
    <script>
      function carregar() {
        valor = confirm("Por favor, clique em
qualquer botão!");
        if (valor) {
          alert("Clicou no botão OK!");
        } else {
          alert("Clicou no botão CAN-
CEL!");
        }
      }
    </script>
  </head>
  <body onload="carregar();">
  </body>
</html>
```

No código apresentado é criado uma **função carregar()** que é executada quando a página é carregada. Ou seja, o evento **onload** carrega a função e executa uma caixa de diálogo para o usuário clicar em uma das opções, **OK** ou **CANCEL**.

Outra forma de interagir com o usuário é solicitar que seja informado algum dado e isso pode ser feito com o uso do comando prompt da linguagem JavaScript. Veja o exemplo apresentado no CÓDIGO 2.33.

CÓDIGO 2.33 – Uso do comando Prompt

```
<!DOCTYPE html>
<html>
  <head>
    <title>window.prompt</title>
    <meta charset="utf-8"/>
    <script>
      valor = prompt("Qual o seu nome!","nome");
      alert(valor);
    </script>
  </head>
  <body>
  </body>
</html>
```

As caixas de diálogo permitem interação do usuário com a página web apresentando informação ou solicitando algum dado a ser processado pelo site.

2.5.7 Eventos

As caixas de diálogo apresentam alguma interatividade com o usuário que navega na página Web.

No entanto, um outro recurso muito importante para a linguagem JavaScript são os eventos que permitem a interação com os elementos de um documento HTML.

Por exemplo, ao clicar em um botão da página é possível acionar uma função criada na linguagem JavaScript e efetuar algum tipo de cálculo, sendo o cálculo apresentado na tela.

Ou então, com base em uma série de dados informados em caixas de texto (inputs html) realizar algum tipo de cálculo após clicar em um botão na página.

Os eventos permitem a interatividade do código JavaScript com os documentos HTML e existem vários tipos de eventos que podem ser associados aos elementos, por exemplo: *onload*, *onblur*, *onchange*, *onfocus*, *onkeydown*, *onkeyup*, *onmouseover*, *onmouseout*, *onclick*, *ondblclick*, *onunload*, *onresize*, entre outros.

2.5.8 Prática

Na prática do tópico da linguagem JavaScript vai ser criada a página do “**Fale Conosco**” do exemplo do Sistema de Gerenciamento de Notícias. Sendo assim, o desenvolvedor vai ser guiado a construir um formulário com os campos nome, assunto e mensagem. A ideia é quando o usuário clicar no botão para enviar uma mensagem, dispare um alerta caso os campos estejam com o valor em branco.

A ideia dessa prática é mostra como a linguagem JavaScript pode ajudar o desenvolvedor a criar validações ou outro tipo de interação.

Entretanto, vale apresentar nesta prática um comando ainda não comentado. A linguagem JavaScript permite que o desenvolvedor crie algumas validações no documento HTML. Para ter acesso aos campos de um formulário por exemplo, o desenvolvedor pode recorrer ao **DOM (Document Object Model)** que permite acessar todos os elementos de uma página Web.

Na linguagem JavaScript pode ser usado o comando **document.getElementById** para recuperar o valor de um campo de um formulário e atribuir o valor deste campo a uma variável local e na sequência realizar as validações necessárias.

O comando **document.getElementById**(*nome do valor atribuído ao id do elemento*) recupera o objeto input (tag HTML mencionada anteriormente) de um determinado ID. Assim, para recuperar o valor deste campo input precisa complementar o comando o `document.getElementById(nomeID).value`.

O formulário criado fica com o *layout* igual ao da Figura 17, considerando a visualização no Firefox.

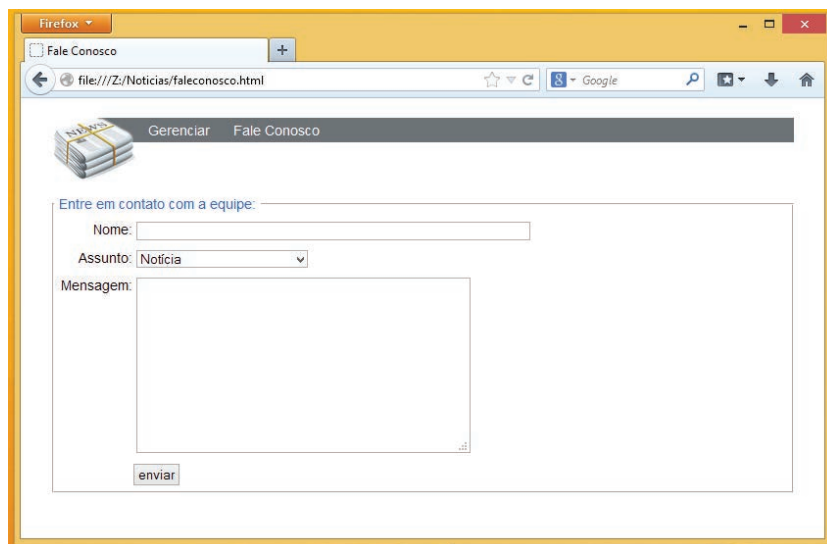


Figura 17 - Formulário “Fale Conosco”

Para gerar a página apresentada na Figura 17 foi criado o código **HTML**, **CSS** e **JavaScript** apresentado a seguir.

arquivo: faleconosco.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Fale Conosco</title>
    <!--Garante o suporte correto do HTML5 para o
Internet Explorer-->
    <!--[if lt IE 9]>
      <script src="http://html5shiv.googlecode.com/
svn/trunk/html5.js"></script>
    <![endif]-->

    <link rel="stylesheet" href="_css/estilo.css"
type="text/css"/>
    <!-- Código JavaScript para obrigar que o usu-
ário só informe algum valor nos campos -->
    <script>
      /*
          A função validar tem como corpo do
código duas variáveis locais com o nome de nome e mensagem.
          Cada variável recebe o valor
do formulário, para isso é usado o comando document.
getElementById(nome do ID do campo) seguido do ponto value.

          Para realizar a comparação é usa-
do a estrutura condicional IF e o operador de comparação
igual == .
      */
      function validar() {
        var nome = document.
getElementById("nome").value;
        var mensagem = document.
getElementById("mensagem").value;
        if (nome == "") {
          alert("Deve ser informado um
valor para o nome!");
        } else if (mensagem == "") {
          alert("Deve ser informado um
valor para a mensagem!");
        }
      }
    </script>
  </head>
  <body>
```

```

        <header>
            <span id="logotipo"></span>
            <nav>
                <ul id="menu">
                    <li><a href="#">Gerenciar</
a></li>
                    <li><a href="faleconosco.
html">Fale Conosco</a></li>
                </ul>
            </nav>
        </header>
        <section>
            <article>
                <form action="#" method="POST">
                    <fieldset>
                        <legend>Entre em conta-
to com a equipe: </legend>
                        <p>
                            <label
for="nome">Nome:</label>
                            <input type="text"
id="nome" name="nome" class="texto" />
                        </p>
                        <p>
                            <label
for="assunto">Assunto:</label>
                            <select
name="assunto">
                                <option
selected>Notícia</option>
                                <option>Solicitar retirada de notícia</option>
                            </select>
                        </p>
                        <p>
                            <label
for="mensagem">Mensagem:</label>
                            <textarea
class="textarea" name="mensagem" rows="10" cols="40"
id="mensagem"></textarea>
                        </p>
                        <p><label><label><input
type="submit" value="enviar" onclick="validar();" /></p>
                    </fieldset>
                </form>
            </article>
        </section>
    </body>
</html>

```


arquivo: estilo.css

```
/*
Define a margem e o espaçamento com o valor igual a zero.

A propriedade margin refere-se a margem e ao aplicar o
valor igual a zero tanto no topo, embaixo, lado direito e
esquerdo da imagem a margem é igual a zero.

A propriedade padding define um espaçamento do conteúdo
com as bordas ao retorno do elemento. Neste caso, quando
definido 0 significa que todos os elementos tem o padding
tanto em cima, abaixo, lado direito e lado esquerdo igual
a zero.
*/
* {
    margin:0;
    padding:0;
}

/*
Define a fonte da página em 88% e fonte igual a Arial, Hel-
vetica e Sans-serif
*/
body {
    font:88% Arial, Helvetica, sans-serif;
}

/*
A propriedade text-decoration pertencente a tag "a" que
define um link no documento HTML, quando atribuída com o
valor none retira o sublinhado.
*/
a {
    text-decoration: none;
}

/*
A propriedade list-style-type retira os bullets de uma
lista. Entende-se por bullets o desenho ou símbolo que fica
ao lado esquerdo de uma lista.
*/
ul,ol {
    list-style-type:none;
}

/*
A propriedade width define a largura do cabeçalho represen-
tado pela tag "header" do HTML5 em 757px de largura.
```

A propriedade `margin` define as margens desse elemento. Entende-se margem como sendo o espaçamento entre o topo, lado direito, lado esquerdo e abaixo de um elemento. Quando a margem está automática significa que o conteúdo fica centralizado na página.

```
*/  
header {  
    width: 757px;  
    margin: auto;  
}
```

/*
A propriedade `background-color` define a cor de fundo do elemento `nav` em cinza. Assim, as cores em folha de estilo (CSS) são definidas em Hexadecimais.

A propriedade `height` define a altura em 25px, dessa forma define uma espessura a barra criada em folha de estilo.

A propriedade `margin-top` define o valor de 20px, ou seja está à 20px em relação ao topo do navegador neste caso.

```
*/  
nav {  
    background-color: #6d7276;  
    height: 25px;  
    margin-top: 20px;  
}
```

/*
A propriedade `float` define o alinhamento do elemento que está com o ID `#logotipo`. Assim, quando o valor da propriedade é igual a `left` indica que o elemento é lançado à esquerda.

```
*/  
#logotipo {  
    float: left;  
}
```

/*
Uma lista é feita por lista de itens e cada item deve ser alinhado à esquerda. Quando a propriedade `left` é atribuída ao elemento `li` significa esse comportamento.

```
*/  
#menu li {  
    float:left;  
}
```

```
/*
```

Quando encontrar um ID definido como #menu e que tenha a tag a é aplicado as seguintes regras:

- Atribuir um padding de 5px margem top;
- Atribuir um padding de 5px margem bottom;
- Atribuir um padding de 12px margem right;
- Atribuir um padding de 12px margem left;
- Atribuir a cor em branco;
- Atribuir a aparência em bloco;

```
*/
```

```
#menu a {  
    padding: 5px 12px;  
    color: #FFFFFF;  
    display: block;  
}
```

```
/*
```

Quando o usuário passa o mouse em cima do link aciona a propriedade hover. Assim, a cor da fonte deve ser preta (#000000), a cor de fundo branca (#FFFFFF) e deve ser apresentado como bloco (display: block). */

```
#menu a: hover {  
    background-color: #FFFFFF;  
    color: #000000;  
    display: block;  
}
```

```
/*
```

Configurando a posição dos campos de entrada do Texto

```
*/
```

```
/*
```

A propriedade width define a largura da tag section em 757px.

A propriedade margin define as margens esquerda, direita, topo e abaixo com valores automáticos.

A propriedade margin-top define a margem do topo igual a 50px.

```
*/
```

```
section {  
    width: 757px;  
    margin: auto;  
    margin-top: 50px;  
}
```

```
/*
```

A propriedade padding define um espaçamento do conteúdo com as bordas ao retorno do elemento. Neste caso, quando

definido 5 significa que o elemento tem o padding tanto em cima, abaixo, lado direito e lado esquerdo igual a cinco.

```
*/  
p {  
    padding: 5px;  
}
```

/*

A propriedade text-align define o alinhamento do texto para direita, esquerda e centro. No contexto que está sendo tratado o texto está sendo alinhado a direita.

A propriedade width estabelece a largura, neste caso está sendo definido o tamanho de 75px do elemento label.

A propriedade vertical-align define que o conteúdo deve ser alinhado no topo do elemento ou alinhado na parte inferior do elemento. No exemplo, o texto está alinhado verticalmente ao topo.

A propriedade display simula o comportamento de um elemento em relação a forma como o elemento deve se comportar. Ou seja, o elemento label é um elemento in-line não provocando quebras de linha aos elementos que seguem o elemento label. No entanto para conseguir aplicar o tamanho width precisa ser definido a propriedade inline-block.

```
*/  
label {  
    text-align: right;  
    width: 75px;  
    vertical-align: top;  
    display: inline-block;  
}
```

/*

A propriedade width define a largura de 400px de uma tag input que tenha uma classe definida como sendo texto.

A propriedade font-size modifica o tamanho da fonte presente no elemento input. A unidade em está relacionado ao tamanho da letra M em maiúsculo sendo variável assim como a definição em porcentagem.

```
*/  
input.texto {  
    font-size:1em;  
    width: 400px;  
}
```

/*

A propriedade font-size modifica o tamanho da fonte presente no elemento input. A unidade em está relacionado ao tamanho da letra M em maiúsculo sendo variável assim como a definição em porcentagem.

```
*/  
input.textarea {  
    font-size: 1em;  
}
```

```
/*
```

A propriedade padding define um espaçamento do conteúdo com as bordas ao retorno do elemento. Neste caso, quando definido 5 significa que todos os elementos tem o padding tanto em cima, abaixo, lado direito e lado esquerdo igual a cinco.

A propriedade color define a cor do texto pertencente ao elemento legend em azul.

```
*/  
legend {  
    padding: 5px;  
    color: #006AD6;  
}
```

Ao clicar no botão enviar apresenta mensagens ao usuário solicitando que seja informado algum dado nos campos nome e mensagem, veja um exemplo na Figura 18.

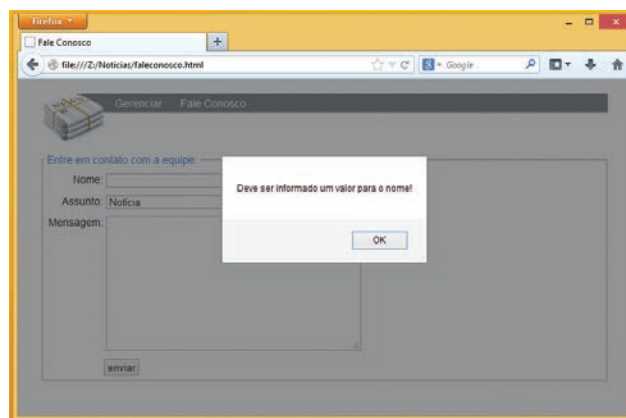


Figura 18 - Mensagem solicitando que dado seja informado

2.6 Considerações finais

Essa unidade introduziu o HTML e o uso de formulários para permitir a interação entre aplicações clientes e servidores Web. Foram abordados assuntos referentes à criação, manipulação e alteração de páginas com HTML, a manipulação da interface gráfica utilizando **Folhas de Estilos (CSS)** e o uso de **JavaScript** para adicionar comportamentos a documentos HTML.

2.6.1 Estudos complementares

DEITEL, H. M.; DEITEL, P.J. – Java Como Programar. 6ª. Edição. Editora Pearson-Prentice Hall, 2005.

GONÇALVES, E. – Desenvolvendo Aplicações Web com NetBeans IDE 5.5 - Editora Ciência Moderna, 2007.

FIELDS, D.K.; KOLB, M.A. – Desenvolvendo na Web com JavaServer Pages – Editora Ciência Moderna, 2000.

SERSON, R.R. – Certificação Java 5. Brasport Livros e Multimidia Ltda, 2006.

UNIDADE 3

Servlet

Servlet

É um componente do lado servidor que gera dados HTML e XML para a camada de apresentação de um aplicativo Web.

O Servlet é basicamente uma classe na linguagem de programação Java que dinamicamente processa requisições e respostas, proporcionando dessa maneira novos recursos aos servidores.

O fato de serem escritos em Java, tornam os Servlets independentes de plataforma, podendo ser criados, compilados e executados em plataformas diferentes.

Servlets possuem estrutura orientada a objetos, o que aumenta a modularização do sistema e são capazes de trabalhar com servidores *multithread* e de tratar *cookies* com muita segurança. A Figura 19 exemplifica o funcionamento dos servlets.

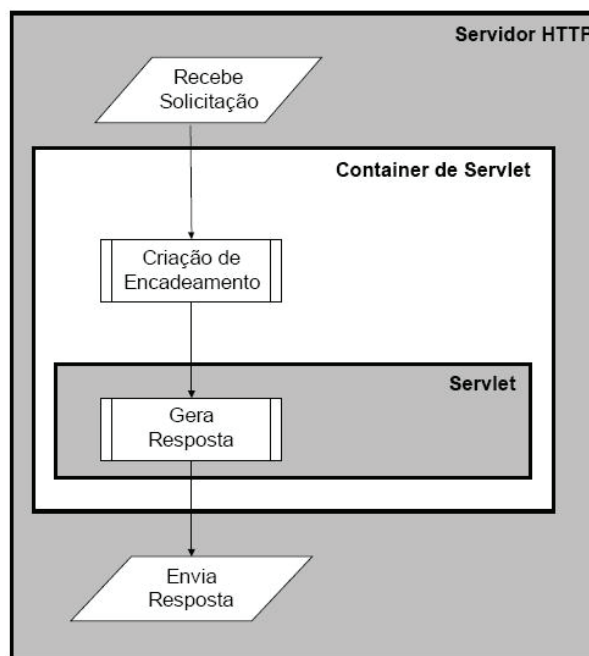


Figura 19 - Funcionamento dos Servlets

Em termos de desempenho os Servlets são carregados apenas uma vez e para cada nova requisição o Servlet gera uma nova *thread*. Isso otimiza o uso da memória e melhora sensivelmente o desempenho, se comparado com outras tecnologias que a cada nova requisição carrega uma cópia completa do código responsável em tratar tal solicitação.

O método `init()` do Servlet, ocorre apenas na primeira vez que a classe é carregada. É geralmente no método `init()` que, por exemplo, estabelecemos uma conexão ao Banco de Dados.

Cada uma das *thread* geradas poderá usar a mesma conexão com o Banco de Dados aberto no método `init()`. O tratamento realizado aumenta em muito o desempenho (tempo de processamento da resposta usando Servlet). Os Servlets podem rodar em qualquer plataforma sem serem reescritos ou até mesmo compilados.

3.1 Primeiras Palavras

Essa unidade tem o objetivo de apresentar os Servlets que são basicamente componentes do lado servidor que geram dados HTML e XML para a camada de apresentação de um aplicativo Web. Sendo assim, os assuntos abordados referem-se à estrutura básica do Servlet, seus principais métodos e para finalizar a Unidade 3 é apresentado um exemplo completo explorando as funcionalidades usando a IDE NetBeans.

3.2 Estrutura Básica do Servlet

Servlet é uma subclasse de `HttpServlet` com a sobreposição dos métodos:

- `service` - trata as requisições com qualquer método;
- `doGet` - trata as requisições com o método GET;
- `doPost` - trata as requisições com o método POST;
- `init` e `destroy` - alocar e liberar recursos disponíveis durante a vida do Servlet, como conexão de Banco de Dados e outros objetos; e
- `getServletInfo` - para retornar informações sobre o próprio Servlet.

Método `init()`

O método `init()` é responsável por iniciar o Servlet e os logs de inicialização. A principal característica deste método é ser carregado apenas a primeira vez que o Servlet executa, ficando depois alocado na memória do servidor.

Nenhuma requisição é processada até que a execução deste método seja concluída. Assim, o método `init()` tem a característica de ser executado apenas na primeira vez em que o Servlet é executado.

Geralmente o método `init()` é utilizado para fazer a conexão ao Banco de Dados, garantindo que exista apenas uma conexão ao Banco de Dados e de que todas as requisições dos clientes usem esta conexão para o acesso aos dados.

Veja a declaração do método `init()` e parâmetros apresentado no CÓDIGO 3.1 a seguir.

CÓDIGO 3.1 – Método `init()`

```
public void init (ServletConfig ConfigInicial) throws ServletException {
    super.init(ConfigInicial);
    // Demais inicializações
}
```

Método `destroy()`

O método `destroy()` tem como função finalizar o Servlet, liberando os recursos alocados e destruindo os logs do Servlet. Sendo assim, pode ser declarado neste método o fechamento da conexão ao Banco de Dados.

O método `destroy()`, assim como o `init()`, é chamado apenas uma vez e automaticamente pelo serviço de rede a cada vez em que o Servlet for removido da memória. Para este método ser chamado novamente é preciso que o Servlet seja recarregado.

Veja a declaração do método `destroy` apresentado no CÓDIGO 3.2 a seguir.

CÓDIGO 3.2 – Método `destroy()`

```
public void destroy( ) {
    // ... }
```

Métodos `doGet()` e `doPost()`

O método `doGet()` e `doPost()` do Servlet tratam as requisições feitas considerando a forma como um Servlet é requisitado, ou seja, usando GET ou POST.

Os dados do formulário são enviados através de uma conexão HTTP para o Servlet que recebe e responde a requisição.

A classe `HttpServlet` possui vários métodos para tratar os vários tipos de requisição (POST, GET). Para formulários com requisições do tipo POST pode-se utilizar o método `doPost(HttpServletRequest, HttpServletResponse)`.

Para formulários com requisições do tipo GET pode-se utilizar o método `doGet(HttpServletRequest, HttpServletResponse)` e assim por diante.

Veja a declaração do método `doGet()` e `doPost()`, juntamente com os possíveis parâmetros apresentado no CÓDIGO 3.3 a seguir.

CÓDIGO 3.3 – `doGet()` e `doPost()`

```
public void doGet(HttpServletRequest Req,
    HttpServletResponse Res) throws IOException,
    ServletException {
    // ... }
public void doPost(HttpServletRequest Req,
    HttpServletResponse Res) throws IOException,
    ServletException {
    // ... }
```

Método `service()`

O método `service()` aceita parâmetros da conexão HTTP e possui dois parâmetros que são `HttpServletRequest` e `HttpServletResponse`.

Assim, o parâmetro `HttpServletRequest` contém informações sobre a requisição, incluindo os parâmetros fornecidos pelo cliente. Já o parâmetro `HttpServletResponse` é usado para retornar as informações para o cliente.

O método não é executado até que o método `init()` seja finalizado.

Veja a declaração do método `service()`, juntamente com os parâmetros `HttpServletRequest` e `HttpServletResponse` apresentados a seguir no CÓDIGO 3.4.

CÓDIGO 3.4 – Método `service()`

```
public void service (HttpServletRequest Req,
    HttpServletResponse Res) throws ServletException,
    IOException {
    // ...
}
```

Métodos `doGet()`, `doPost()` e `service()`

Todos os métodos mencionados tais como `doGet()`, `doPost()` e `service()` recebem dois parâmetros que são objetos das classes: `HttpServletRequest` e `HttpServletResponse`.

O acesso às informações enviadas pelo cliente é realizada através da interface `HttpServletRequest` pelo `Servlet`. Por exemplo: os dados do formulário, o tipo do método HTTP usado (GET, POST), os cookies de um computador, etc.

A interface `HttpServletResponse` por sua vez permite ao Servlet enviar os dados para o cliente, manipular o protocolo HTTP, especificar informações de cabeçalho da conexão.

Além disso, a interface `HttpServletResponse` possui métodos que permitem adicionar um cookie no cliente ou redirecioná-lo para outro endereço.

Vale complementar que a *interface `HttpServletRequest`* é uma extensão da *interface `ServletRequest`* e o mesmo acontece com a *interface `HttpServletResponse`* que estende a *interface `ServletResponse`*.

Métodos da Interface `HttpServletRequest`

Apresentamos agora uma tabela com os principais métodos da interface `HttpServletRequest`.

<code>String getParameter(String name)</code>	Obtém o valor de um parâmetro (name) enviado ao Servlet como parte de uma solicitação GET ou POST.
<code>Enumeration getParameterNames()</code>	Retorna o nome de todos os parâmetros enviados ao Servlet como parte de uma solicitação POST.
<code>Cookie [] getCookies()</code>	Retorna um array de objetos <code>Cookie</code> armazenados no cliente pelo Servidor. Objetos <code>Cookies</code> podem ser utilizados para identificar unicamente os clientes do Servlet.
<code>HttpSession getSession (boolean create)</code>	Retorna um objeto <code>HttpSession</code> associado com a sessão atual.
<code>String getLocalName ()</code>	Obtém o nome de host em que a solicitação foi recebida.
<code>String getLocalAddr()</code>	Obtém o endereço IP em que a solicitação foi recebida.
<code>int getLocalPort ()</code>	Obtém o número da porta do IP em que a solicitação foi recebida.
<code>void addCookie (Cookie cookie)</code>	Adiciona um cookie ao cabeçalho da resposta ao cliente.
<code>ServletOutputStream ()</code>	Obtém um fluxo de saída baseado em bytes para enviar dados binários.
<code>PrintWriter getWriter()</code>	Obtém um fluxo de saída baseado em caracteres para enviar dados de texto (normalmente em formato HTML) ao cliente.

3.3 Tratando caracteres especiais

Nas aplicações a linguagem empregada para descrever os campos, as mensagens e as ações são sempre criadas de acordo com o idioma do cliente.

Em alguns casos, o sistema tem que suportar a internacionalização de idioma, ou seja, habilitar o suporte para mais de uma linguagem. Por exemplo: português, espanhol e inglês americano.

No idioma da língua portuguesa brasileira, tem-se a presença de caracteres acentuados e isso pode representar um problema ao exibir dados recuperados de banco de dados ou repassados por Servlet como os exemplos desta Unidade.

O desenvolvedor precisa empregar corretamente o cabeçalho adequado das páginas que vão criar para evitar tipos de problemas, tais como apresentar caracteres estranhos no lugares de palavras acentuadas.

Sendo assim, preste atenção na definição da meta tag que define o encoding da página que pode ser UTF-8 ou ISO-8859-1. E nos Servlet desenvolvido aplique o tipo de encoding adequado para tratar os atributos recebidos e a nova página a ser criada pelo Servlet.

Uma possível solução para tratar palavras acentuadas é exemplificada no código a seguir.

O CÓDIGO 3.5 apresenta uma página HTML5 que manda palavras acentuadas digitadas por um usuário a um Servlet denominado “Acentuacao”. Observe que o código da página HTML tem a especificação da meta tag de charset definido para UTF-8.

CÓDIGO 3.5 – Acentuação

```
<!DOCTYPE html>
<html>
<head>
    <title>Informe alguma palavra com acento</title>
    <meta charset="UTF-8">
</head>
<body>
    <h1>SOMA</h1>
    <form action="http://localhost:8080/SomaServlet/ser-
vlet/Acentuacao" method="POST">
        Palavra: <input type="text" name="palavra"><br/>
        <input type="submit" value="OK">
    </form>
</body>
</html>
```

O Servlet desenvolvido recebe o campo palavra e exibe em uma nova página HTML5. No código do método doPost() é realizada uma conversão do campo submetido ao Servlet e também é definido o tipo de charset da página gerada pelo Servlet.

Observe no CÓDIGO 3.6 as linhas implementadas para sanar o problema relacionado a exibição de caracteres especiais.

CÓDIGO 3.6 – Caracteres especiais em Servlet

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    int numero1 = Integer.parseInt(request.getParameter("num1"));
    int numero2 = Integer.parseInt(request.getParameter("num2"));
    int result = adicao(numero1, numero2);

    / * *
    O método setContentType aplicado no atributo response
    do tipo HttpServletResponse converte o texto a ser
    apresentado no navegador para o padrão de codificação
    UTF-8.
    * */
    response.setContentType("text/html;charset=UTF-8");

    /** Uma variável local do tipo String denominado "palavra"
    recebe o valor do campo submetido ao Servlet "Acentuacao".
    Uma variável local do tipo array de byte denominado "bytes"
    recebe o byte da variável palavra no padrão charset brasileiro
    ISO-8859-1.
    A variável local do tipo String "palavra" criada anteriormente,
    recebe uma nova instância de String considerando o valor de byte
    e o tipo de charset UTF-8.
    Este trecho de código trata o campo submetido ao Servlet para
    que seja apresentado adequadamente considerando os caracteres
    especiais.
    **/
    String palavra = request.getParameter("palavra");
    byte[] bytes = name.getBytes(StandardCharsets.ISO_8859_1);
    palavra = new String(bytes, StandardCharsets.UTF_8);

    PrintWriter saida = response.getWriter();
    saida.println("<!DOCTYPE html>");
}
```



```

saida.println("<html>");
saida.println("<head>"); saida.println("<title>Nome</
title>");
saida.println("<meta http-equiv='Content-Type'
content='text/html; charset=UTF-8'>");
saida.println("</head>"); saida.println("<body>");
saida.println("A palavra é: "+palavra+"</br>"); saida.
println("</body>");
saida.println("</html>");
}

```

3.4 Exemplo de Servlet

A Unidade 3 aborda Servlet e para conseguir compilar o código e executar é necessário que o usuário tenha instalado o Java (<http://www.oracle.com/technetwork/pt/java/javase/downloads/index.html>) e o servidor web Tomcat (<http://tomcat.apache.org/>). Existem várias versões do Java e do Tomcat, recomendamos sempre a mais atual.

O exemplo consiste em criar um formulário contendo dois campos de texto e um botão. E o usuário deve digitar o nome de uma pessoa, digitar uma frase e clicar no botão “Enviar”.

O formulário envia os dados para um Servlet chamado PessoaSrv. O Servlet recebe os dois valores passados pelo formulário, instância um objeto da classe Pessoa e chama o método falar() da classe. O resultado é exibido no navegador do usuário.

O primeiro passo é criar um arquivo chamado pessoa.html e colocá-lo na pasta dsw do diretório de aplicações webapps do Tomcat.

Abra o Bloco de notas e digite o seguinte CÓDIGO 3.7 apresentado a seguir.

CÓDIGO 3.7 – pessoa.html

```

<!DOCTYPE html>
<html>
  <head>
    <title>Pessoa Servlet</title>
    <meta http-equiv="Content-Type" content="text/
html; charset=UTF-8">
  </head>
  <body>
    <form action="PessoaSrv" method="POST">
      Nome: <input type="text" name="txtNome"> <br/>

```

```

        Frase: <input type="text" name="txtFrase"> <br/>
        <input type="submit" value="Enviar">
    </form>
</body>
</html>

```

Salve o arquivo e teste sua execução no navegador, digitando na barra de endereços: <http://localhost:8080/dsw/pessoa.html>.

O formulário a exibido é similar ao apresenta na Figura 20.

The image shows a simple web form. It consists of two text input fields stacked vertically. The first field is labeled 'Nome:' and the second is labeled 'Frase:'. Below these fields is a button with the text 'Enviar'.

Figura 20 - Formulário com nome e frase a ser digitada pelo usuário

Na sequência, crie o arquivo Pessoa.java que contém a classe Pessoa a ser usada pelo Servlet.

Abra o Bloco de notas e digite o CÓDIGO 3.8 como apresentado a seguir.

CÓDIGO 3.8 - Pessoa.java

```

public class Pessoa {
    private String nome;
    public Pessoa() {
    }
    public Pessoa(String nome) {
        this.nome = nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String falar(String frase) {
        return (nome + " falou: " + frase);
    }
}

```

A classe Pessoa possui dois métodos e dois construtores. O método falar() exibe o nome da pessoa e a frase que a pessoa falou.

Salve o arquivo como Pessoa.java (o mesmo nome da classe) no diretório src utilizado no exemplo anterior. Certifique-se que a extensão é mesmo .java.

O próximo passo é criar o Servlet que recebe os dados provenientes do formulário pessoa.html.

Abra o Bloco de notas e digite o seguinte CÓDIGO 3.9 apresentado a seguir.

CÓDIGO 3.9 – PessoaSrv.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PessoaSrv extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<!DOCTYPE html >");
        out.println("<head>");
        out.println("<title>Pessoa Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet PessoaSrv - GET</h1>");

        Pessoa p = new Pessoa();
        p.setNome(request.getParameter("txtNome"));
        out.println(p.falar(request.getParameter("txtFrase")));
        out.println("</body>");
        out.println("</html>");
    } finally {
        out.close();
    }
}

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<!DOCTYPE html >");
        out.println("<head>");
        out.println("<title>Pessoa Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet PessoaSrv - POST</h1>");
        Pessoa p = new Pessoa();
        p.setNome(request.getParameter("txtNome"));
    }
}
```

```

        out.println(p.falar(request.getParameter
("txtFrase")));
        out.println("</body>");
        out.println("</html>");
    } finally {
        out.close();
    }
}

```

Salve o arquivo como PessoaSrv.java (o mesmo nome da classe) no diretório src e certifique se a extensão é mesmo .java.

O próximo passo é compilar servlet PessoaSrv para gerar o arquivo PessoaSrv.class, e também o arquivo Pessoa.java para gerar Pessoa.class.

Para isso, verifique se a variável de ambiente CLASSPATH está configurada corretamente, contendo o seguinte valor:

```

CLASSPATH= C:\Arquivos de programas\Java\jdk1.6.0_11\lib;.;
C:\Arquivos de programas\Apache Software Foundation\Tomcat
6.0\lib\servlet-api.jar

```

Caso a variável CLASSPATH não esteja configurada corretamente, interrompa a leitura deste texto e verifique a configuração das variáveis de ambiente.

Abra o prompt de comando em Iniciar → Programas → Acessórios → Prompt de comando e vá até o diretório src da nossa aplicação:

```

C:\Documents and Settings\latosensu> cd \ C:\> cd "Arquivos
de programas"
C:\Arquivos de programas> cd "Apache Software Foundation" C:\
Arquivos de programas\Apache Software Foundation> cd "Tomcat
6.0" C:\Arquivos de programas\Apache Software Foundation\
Tomcat 6.0> cd webapps C:\Arquivos de programas\Apache Sof-
tware Foundation\Tomcat 6.0\webapps> cd dsw
C:\Arquivos de programas\Apache Software Foundation\Tomcat
6.0\webapps\dsw> cd WEB-INF
C:\Arquivos de programas\Apache Software Foundation\Tom-
cat 6.0\webapps\dsw\WEB-INF> cd src
C:\Arquivos de programas\Apache Software Foundation\Tom-
cat 6.0\webapps\dsw\WEB-INF\src>

```

Estando no diretório src, compile primeiro a classe Pessoa digitando:

```

C:\Arquivos de programas\Apache Software Foundation\Tom-
cat 6.0\webapps\dsw\WEB-INF\src> javac Pessoa.java

```

Aguarde alguns instantes até que a compilação termine. Caso apareça algum erro, verifique o código Java à procura de algum caractere estranho (“Copiar/Colar” às vezes dá algum problema) e repita o passo de compilação.

Agora, compile o servlet PessoaSrv:

```
C:\Arquivos de programas\Apache Software Foundation\Tomcat 6.0\webapps\dsw\WEB-INF\src> javac PessoaSrv.java
```

Aguarde alguns instantes até que a compilação termine. Caso apareça algum erro, verifique o código Java à procura de algum caractere estranho (“Copiar/Colar” às vezes dá algum problema) e repita o passo de compilação.

Verifique o diretório src e veja que os arquivos Pessoa.class e PessoaSrv.class foram criados com sucesso!

O servlet está pronto, e deve ser colocado no diretório apropriado. Mova os dois arquivos Pessoa.class e PessoaSrv.class do diretório src para dentro do diretório classes.

Falta agora fazer o mapeamento do servlet. Para isso, vamos criar o arquivo web.xml. Abra o Bloco de notas e digite o CÓDIGO 3.10 apresentado a seguir.

CÓDIGO 3.10 – web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>PessoaSrv</servlet-name>
    <servlet-class>PessoaSrv</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>PessoaSrv</servlet-name>
    <url-pattern>/PessoaSrv</url-pattern>
  </servlet-mapping>
</web-app>
```

Salve esse arquivo como web.xml dentro do diretório WEB-INF de dsw.

Importante: se o arquivo web.xml já existe, adicione apenas as seguintes linhas no final do arquivo, antes da linha </web-app>:

```
<servlet>
  <servlet-name>PessoaSrv</servlet-name>
```

```
<servlet-class>PessoaSrv</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>PessoaSrv</servlet-name>
  <url-pattern>/PessoaSrv</url-pattern>
</servlet-mapping>
```

Assim, o servlet já está pronto para ser executado!

Digite no navegador: `http://localhost:8080/dsw/pessoa.html`

Digite um nome e uma frase (por exemplo, UAB e “Hello World”) no formulário e clique no botão Enviar. O Servlet é carregado para o servidor, processará a requisição e emitirá a resposta:

```
UAB falou: Hello World
```

Importante: A cada vez que o código compilado do Servlet é alterado, você precisa reiniciar o servidor Tomcat. Isso porque o Servlet, ao ser executado pela primeira vez, é carregado em memória e lá permanece por um bom tempo.

3.5 Exemplo Servlet com NetBeans

Uma outra opção de executar os exemplos é criá-los através de um ambiente de desenvolvimento como o NetBeans (<https://netbeans.org/downloads/>). Para os exemplos deste livro foi utilizado a versão do NetBeans de número 8.02.

A prática se baseia em criar uma aplicação Web composto de um formulário com dois campos de texto, dois botões e um Servlet que recebe o valor contido nos dois campos.

O Servlet deve realizar a soma dos campos e retornar o resultado ao usuário. Logo, a aplicação Web deve ser criada seguindo a seguinte sequência de passos:

PASSO 1

Clique em **Arquivo** → **Novo Projeto**, como apresentado na Figura 21.

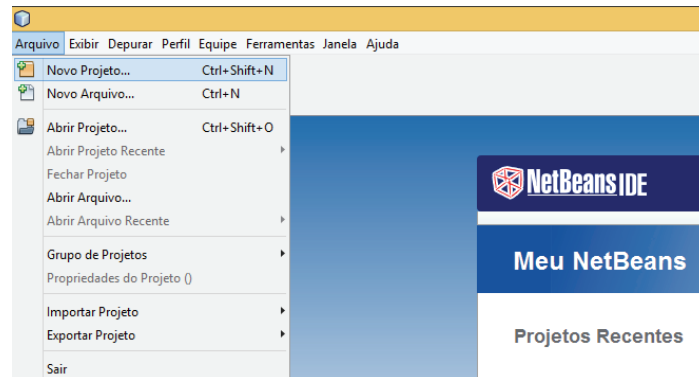


Figura 21 - Novo Projeto

PASSO 2

Na sequência selecione a categoria *Java Web* e defina o projeto como *Aplicação Web*, como apresentado na Figura 22. Depois clique em **Próximo** para continuar o processo de criação.

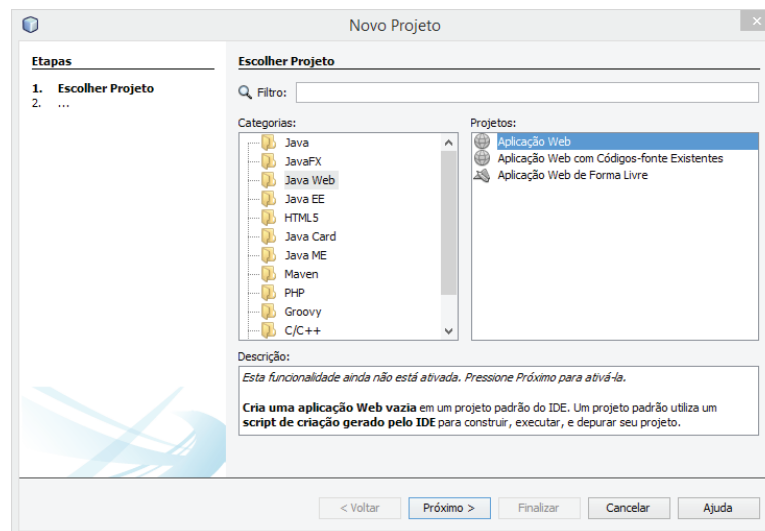


Figura 22 - Seleção do tipo de projeto

PASSO 3

Na tela apresentada na Figura 23, insira o nome do projeto igual SomaServlet e defina um local onde o NetBeans deve criar o projeto. Geralmente é apresentado um local padrão, mas esse local pode ser modificado se for necessário. Após configurado os campos, clique no botão **Próximo**.

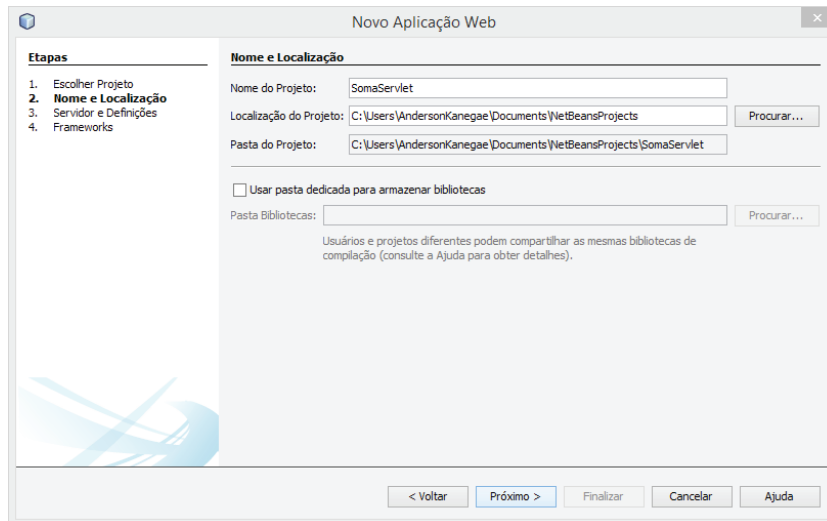


Figura 23 - Definir nome de uma aplicação Web

PASSO 4

Na tela apresentada na Figura 24 é possível configurar o servidor Web. Dessa forma, selecione o servidor Web Tomcat e depois clique no botão **Próximo** para continuar.

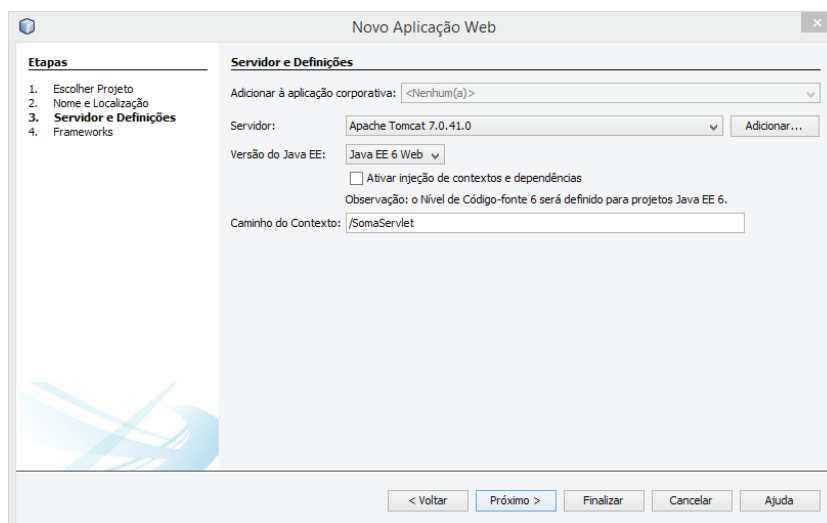


Figura 24 - Selecionar servidor Web

PASSO 5

O NetBeans abre uma tela para selecionar *frameworks*, Figura 25, logo após definir o servidor Web, como apresentado na Figura 24.

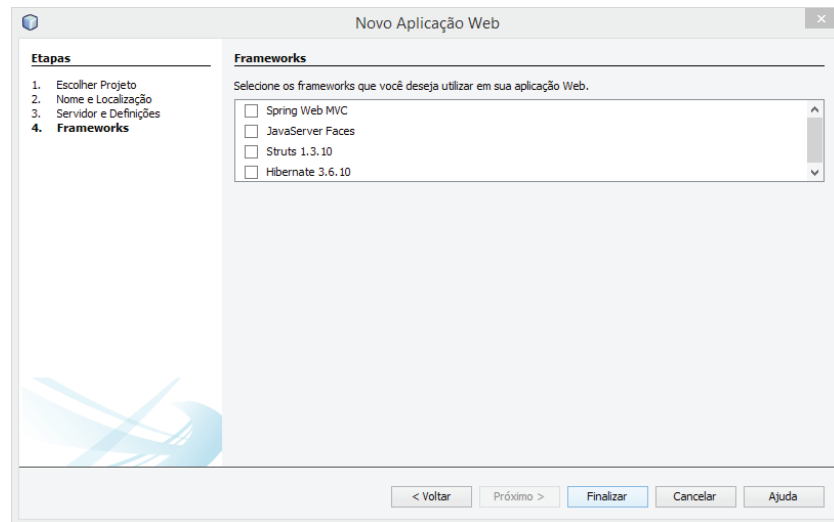


Figura 25 - Selecionar framework

PASSO 6

Para continuar com o desenvolvimento da aplicação Web deve ser criado um documento HTML. Para isso, clique com o botão direito do mouse em **Páginas Web** → **Novo** → **HTML**, como apresentado na Figura 26.

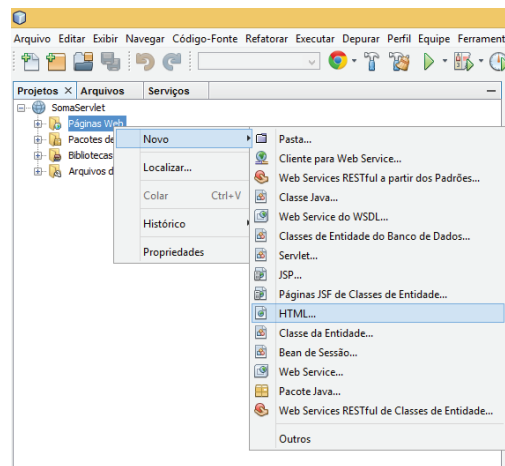


Figura 26 - Criar arquivo HTML

PASSO 7

Ao clicar com o botão direito na opção HTML é aberto uma tela como apresentado na Figura 27. Nesta tela, o desenvolvedor deve preencher o campo “Nome do Arquivo HTML” com o valor Formulario.

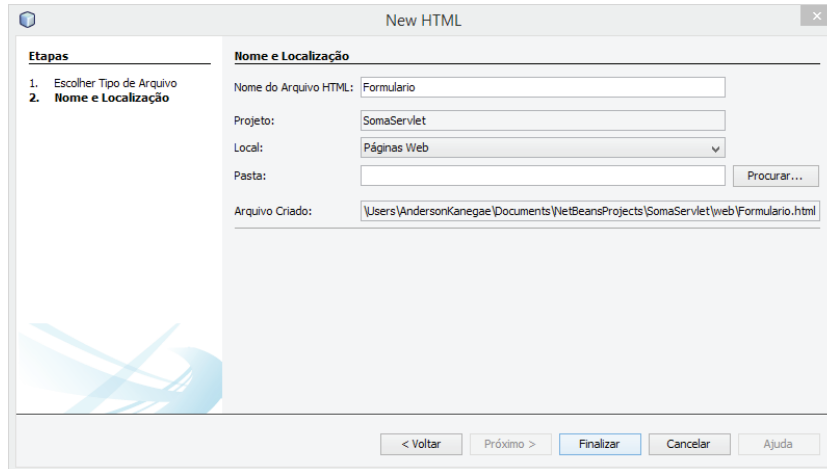


Figura 27 - Definindo nome do arquivo HTML

PASSO 8

Após o **arquivo Formulario.html** ser criado é necessário que o desenvolvedor digite o seguinte código apresentado a seguir:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemplo</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width">
  </head>
  <body>
    <h1>SOMA</h1>
    <form action="http://localhost:8084/SomaServlet/servlet/Soma" method="POST">
      Numero 1: <input type="text" name="num1"><br/>
      Numero 2: <input type="text" name="num2"><br/>
      <input type="reset" value="Limpar">
      <input type="submit" value="OK">
    </form>
  </body>
</html>
```

Observação: Verifique o atributo action do elemento formulário. Neste atributo é especificado caminho do localhost e a porta do Tomcat, verifique qual a porta que foi reservada pelo Sistema Operacional para executar o Tomcat.

PASSO 9

Crie o arquivo **Soma.java**. Para isso, clique com o botão direito do mouse em **Pacotes de Código-fonte** → **Novo** → **Classe Java**, como apresentado na Figura 28.

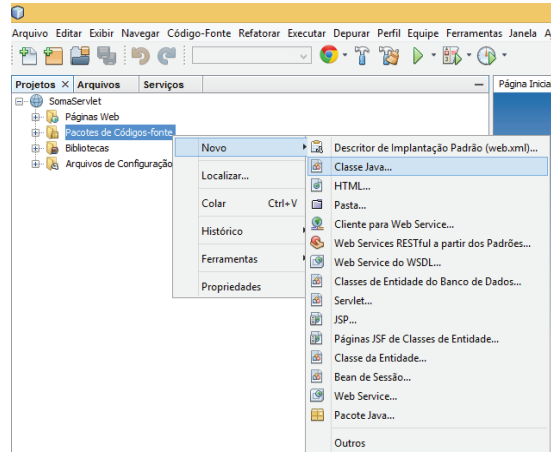


Figura 28 - Criar classe Java

PASSO 10

Após clicar com o botão direito na opção “Classe Java..”, abre-se uma tela como apresentado na Figura 29. Nesta tela deve ser inserido no campo “Nome da Classe” o valor igual a Soma.

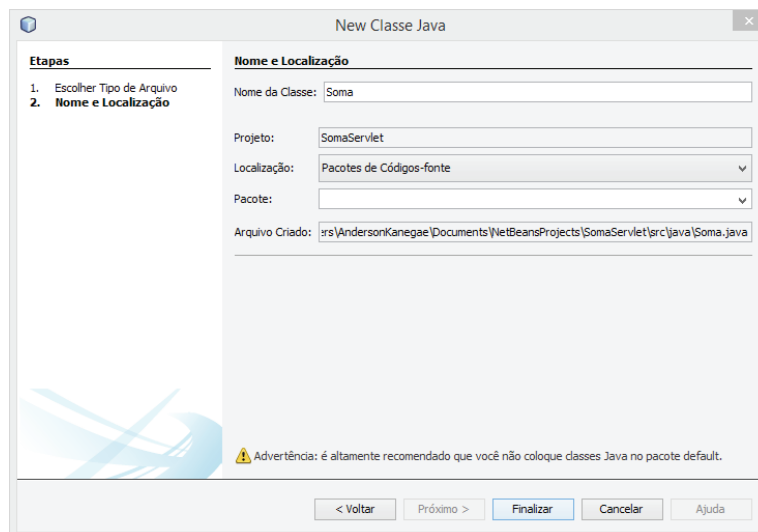


Figura 29 - Definindo nome de uma classe Java

PASSO 11

Após o **arquivo Soma.java** ser criado é necessário que o desenvolvedor digite o seguinte código apresentado a seguir:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Soma extends HttpServlet {

    public void init(ServletConfig ConfInicial) throws ServletException {
        // método chamado para inicializar o servlet
        super.init(ConfInicial);
    }

    public int adicao(int n1, int n2) {
        return (n1 + n2);
    }

    //método chamado para atender uma solicitação de HTTP POST
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        int numero1 = Integer.parseInt(request.getParameter("num1"));
        int numero2 = Integer.parseInt(request.getParameter("num2"));
        int result = adicao(numero1, numero2);
        PrintWriter saida = response.getWriter();
        saida.println("O resultado da soma POST e: " + result);
    }

    //método chamado para atender uma solicitação de HTTP GET
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        int numero1 = Integer.parseInt(request.getParameter("num1"));
        int numero2 = Integer.parseInt(request.getParameter("num2"));
        int result = adicao(numero1, numero2);
        PrintWriter saida = response.getWriter();
        saida.println("O resultado da soma GET e: " + result);
    }

    //método chamado quando o servlet é terminado
    public void destroy() {
```

```

        System.out.println("Fim do servlet");
    }
} // término do servlet

```

PASSO 12

Após criado o **arquivo HTML** e a **classe Java** deve ser criado o arquivo denominado **web.xml** para configurar os Servlets. Para isso, clique com o botão direito do mouse em **WEB-INF** → **Novo** → **Outros** como apresentado na Figura 30.

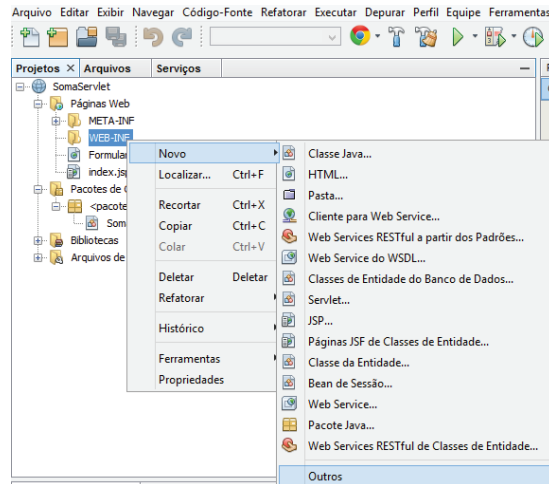


Figura 30 - Adicionar um arquivo web.xml

PASSO 13

Após clicar na opção “Outros” com o botão direito aparece uma tela como apresentado na Figura 31. Assim, selecione a opção **Web** → **Descritor de Implantação Padrão (web.xml)** e clique em **Próximo** para avançar para a próxima etapa.

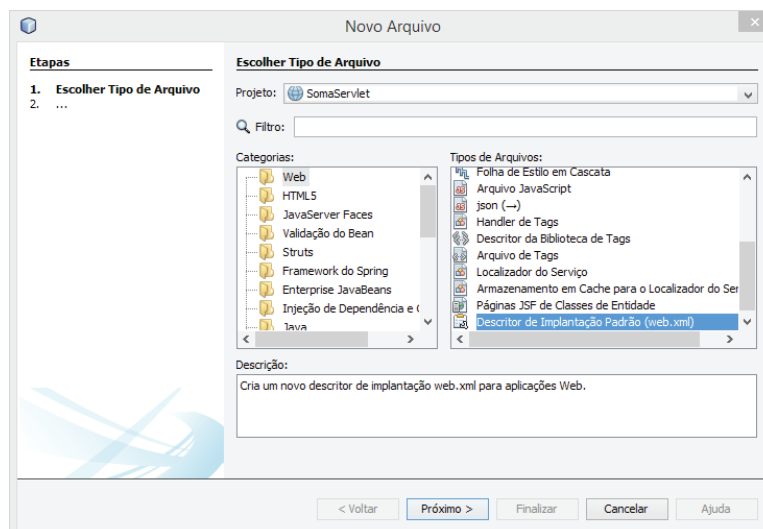


Figura 31 - Arquivo web.xml

PASSO 14

Após selecionado o tipo de arquivo e clicado no botão **Próximo** o NetBeans direciona para uma tela de confirmação apresentado na Figura 32. Confirme os dados e clique em **Finalizar** para criar o arquivo **web.xml**.

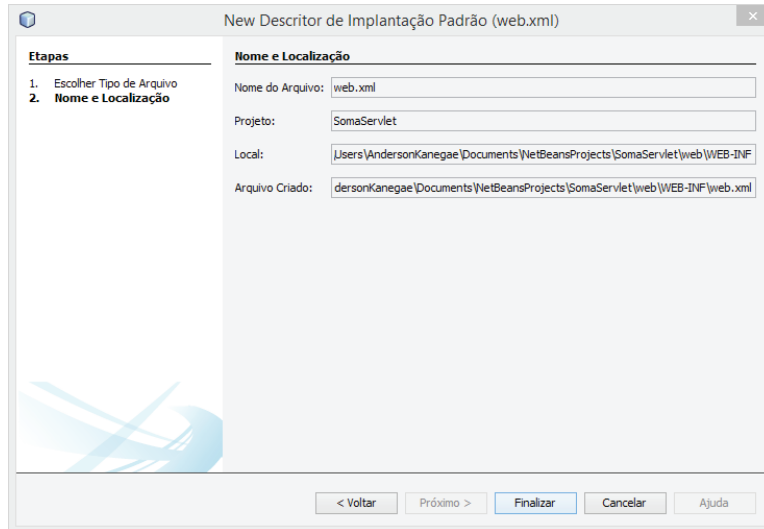


Figura 32 - Finalizando criação de um arquivo web.xml

PASSO 15

Após a criação do **arquivo web.xml** é necessário configurar o Servlet. Para isso, selecione a aba Servlets com o arquivo web.xml aberto e em seguida “Adicionar elemento servlet...”, como apresentado na Figura 33.

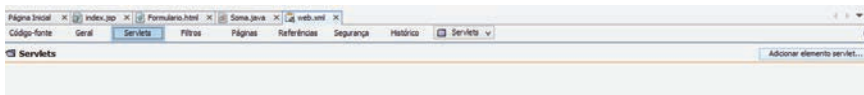


Figura 33 - Configurar o Servlet no web.xml

PASSO 16

Ao clicar na opção “Adicionar elemento servlet...” selecione o Servlet Soma, como apresentado na Figura 34.

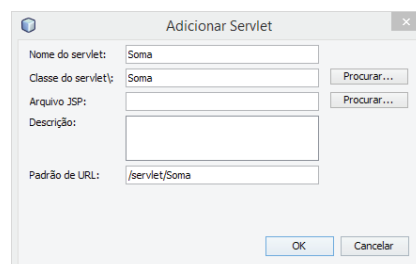


Figura 34 - Adicionando Servlet

PASSO 17

Por fim, antes de executar a aplicação Web criada clique com o botão direito do mouse no nome do projeto e em seguida escolha a opção “Limpar e construir”, como apresentado na Figura 35.

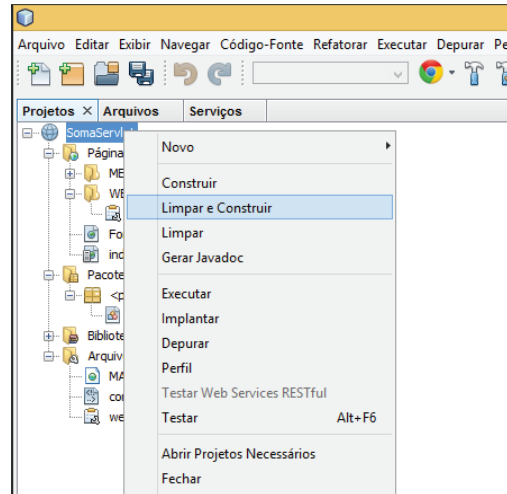


Figura 35 - Preparando aplicação Web

PASSO 18

Para executar o formulário Web responsável em chamar os serviços do Servlet, clique com o botão direito do mouse sobre o **arquivo Formulario.html** e escolha a opção “**Executar Arquivo**”, como apresentado na Figura 36.

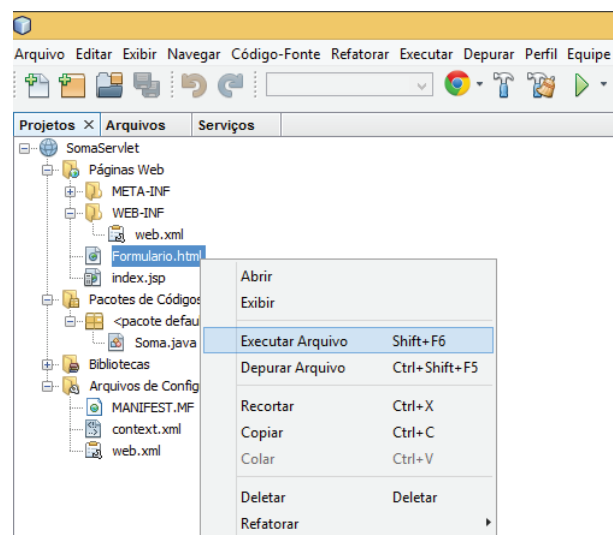


Figura 36 – Executar arquivo Formulario.html

O **arquivo Formulario.html** é executado no navegador Web que foi definido nas preferências do NetBeans. O Formulario.html abre uma página solicitando a inserção de valores para soma, como apresentado na Figura 37.

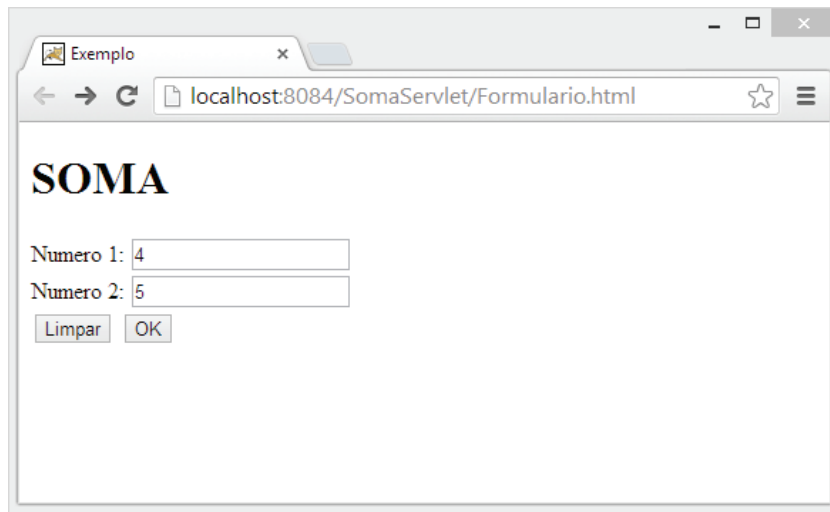


Figura 37 – Visualização do Formulario.html no navegador

Os valores informados são somados ao clicar no botão OK. Assim, o resultado da execução após a requisição feita ao servidor e ao Servlet são exibidos no navegador, como apresentado na Figura 38.

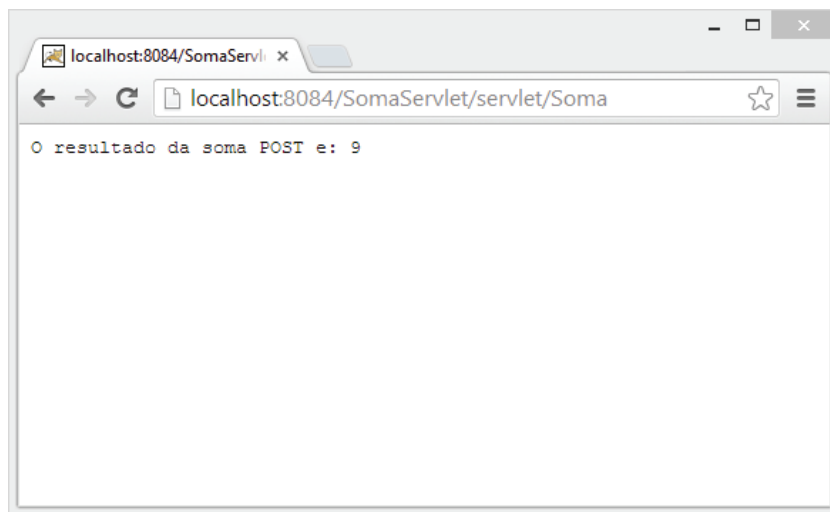


Figura 38 – Resultado da execução do Servlet Soma.java

3.6 Considerações finais

Essa unidade apresentou o conceito de Servlets e seus principais métodos. Foram abordados os aspectos referentes ao desenvolvimento de Servlets utilizando a IDE NetBeans, sua compilação e sua disponibilização no servidor Web. Finalmente os conceitos de Servlets foram explorados através de um exemplo prático.

3.6.1 Estudos complementares

DEITEL, H. M.; DEITEL, P.J. – Java Como Programar. 6ª. Edição. Editora Pearson- Prentice Hall, 2005.

GONÇALVES, E. – Desenvolvendo Aplicações Web com NetBeans IDE 5.5 - Editora Ciência Moderna, 2007.

FIELDS, D.K.; KOLB, M.A. – Desenvolvendo na Web com JavaServer Pages – Editora Ciência Moderna, 2000.

SERSON, R.R. - Certificação Java 5. Brasport Livros e Multimidia Ltda, 2006.

UNIDADE 4

JSP

JSP

Java Server Pages (JSP) é uma tecnologia utilizada no desenvolvimento de aplicações para *Web*, similar às tecnologias Active Server Pages (ASP) da Microsoft ou PHP.

Por ser baseada na linguagem de programação Java, tem a vantagem da portabilidade de plataforma, que permite a sua execução em diversos sistemas operacionais, como o Windows da Microsoft, Unix e Linux. Esta tecnologia permite ao desenvolvedor de páginas para Internet produzir aplicações que acessem o banco de dados, manipulem arquivos no formato texto, capturem informações a partir de formulários e capturem informações sobre o visitante e sobre o servidor.

Uma página criada com a tecnologia JSP, depois de instalada em um servidor de aplicação compatível com a tecnologia Java EE, é transformada em um Servlet. São exemplos de servidor compatível com a tecnologia JSP o Tomcat e o Glassfish.

4.1 Primeiras Palavras

Essa unidade apresenta a tecnologia JSP utilizada no desenvolvimento de aplicações para *Web* e que permite o processamento de linguagens de script no lado servidor para geração de conteúdo dinâmico. Serão abordados nessa unidade o funcionamento do JSP, a estrutura, e os componentes da linguagem. Para complementar o estudo da tecnologia são apresentados dois exemplos completos de páginas dinâmicas utilizando a tecnologia JSP desenvolvidos na IDE NetBeans.

4.2 Funcionamento do JSP

Quando se faz a solicitação de uma página JSP para o servidor, a *engine* do JSP compila a página que foi solicitada e a converte em um servlet que, quando executado, gera a resposta requisitada. A compilação apenas acontece quando ainda não existe o servlet referente à página ou quando o arquivo JSP foi alterado.

O enlace entre uma página JSP e seu correspondente servlet se realiza por meio de duas classes contidas dentro do pacote `javax.servlet.jsp`: `JSPPage` e `HttpJspPage`. Estas são as classes que nos permitem criar a interface compilada do JSP. A Figura 39 exemplifica o funcionamento do JSP.

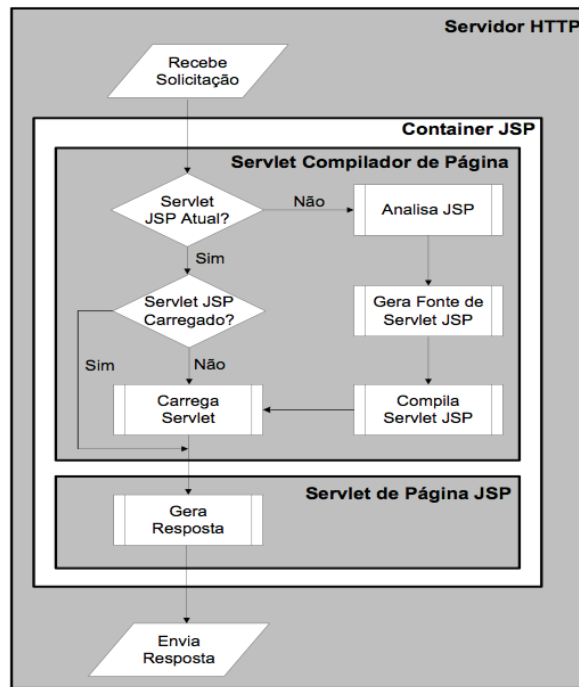


Figura 39 - Funcionamento JSP

4.3 Estrutura e Componentes da Linguagem

- **Diretivas:** são mensagens para a *engine* JSP. Informam a *engine* JSP o que fazer com a página. São sempre encapsuladas por `<%@ ... %>`. Alguns exemplos de diretivas do JSP são `page` e `include`.
- **Declarações:** permitem definir variáveis no nível de página para salvar informações ou definir métodos de suporte que o resto de uma página JSP pode necessitar.
- **Expressões:** o resultado das expressões em JSP é diretamente convertido em uma string e incluído dentro da página de saída.
- **Scriptlets:** fragmentos de código JSP, encapsulados por `<% ... %>`. Este código Java roda quando um request é servido pela página JSP. Você pode ter código Java dentro de um scriptlet.
- **Comentários:** encapsulados por `<%-- ... --%>`.
- **Objetos:** podem ser criados implicitamente pelas diretivas, explicitamente através de ações ou usando `script`. Os objetos Java dentro de uma página JSP têm um escopo associado a eles. Os escopos que podem ser associados a um objeto dentro de uma página JSP são:

- *Application*: objetos acessíveis para todas as páginas da aplicação.
- *Session*: objetos acessíveis apenas nas páginas de mesma seção em que foram criados.
- *Request*: objetos acessíveis nas páginas processando a requisição onde eles foram criados.
- *Page*: objetos acessíveis apenas nas páginas onde eles foram criados.

JSP possui objetos implícitos, que podem ser usados sem a necessidade de serem criados. São definidos dentro da API de Servlet. A tabela abaixo ilustra esses objetos e suas funcionalidades.

Request	Representa o HttpServletRequest, engatilhando a invocação do serviço. Faz a requisição de informações.
Response	Representa o HttpServletResponse para a requisição. Envia uma resposta na página.
PageContext	Encapsula características dependentes de implementação em PageContext
Application	Representa o ServletContext obtido do objeto de configuração do servlet.
Out	Objeto JspWriter que escreve no fluxo de saída (na página)
Config	Representa o ServletConfig para o JSP
Page	Sinônimo para o operador "this", é um HttpJspPage
Session	Um HttpSession. Associado a sessão de usuário. Escopo de session. Por default, todas as páginas JSP participam de uma seção HTTP. Uma seção é um bom lugar para armazenar objetos e dados que devem ser acessados por várias páginas e/ou por vários servlets pelo usuário. A seção é identificada por um session ID que é armazenado no browser do usuário como um cookie.
Exception	Um objeto Throwable que resultou em um erro de página

4.3.1 Exemplo 1

O primeiro exemplo de aplicação Web com JSP baseia-se em criar algumas páginas JSP. Para isso, siga as sequências de passos para criação de cada página JSP explicada a seguir:

Página: tags.jsp

PASSO 1

Inicialize o NetBeans.

PASSO 2

Crie um “Novo Projeto...”, para isso clique na opção **Arquivo** → **Novo Projeto** como apresentado na Figura 40.

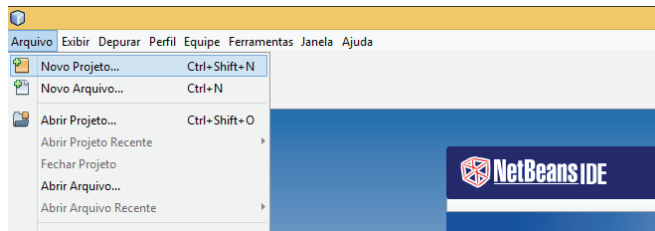


Figura 40 Novo projeto

PASSO 3

Na tela apresentada na Figura 41 o desenvolvedor deve escolher a **categoria** do projeto igual a Java Web e o **tipo de projeto** igual a Aplicação Web. Na sequência, clique no botão **Próximo**.

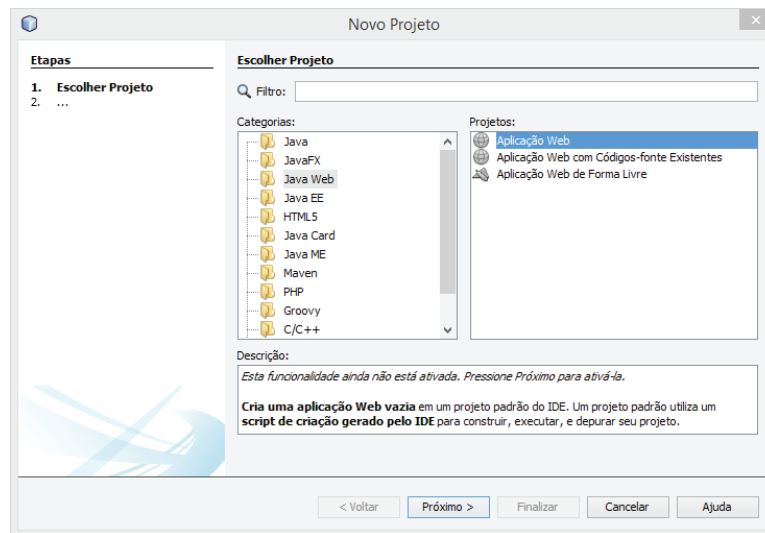


Figura 41 Selecionar tipo de projeto

PASSO 4

Após definir as características do novo projeto deve ser informado o nome do projeto. Na tela apresentada na Figura 42, deve ser inserido o nome do projeto igual a **tagsJSP**. Na sequência, clique no botão **Próximo**.

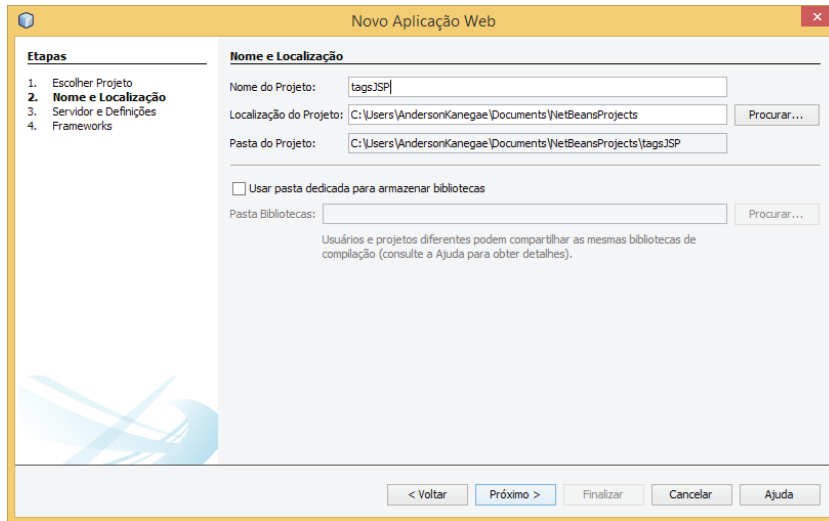


Figura 42 - Definir nome do projeto

PASSO 5

O NetBeans direciona a uma tela em que deve ser selecionado o servidor a ser usado na aplicação Web. Escolha o Tomcat na lista e clique no botão **Próximo**, como apresentado na Figura 43.

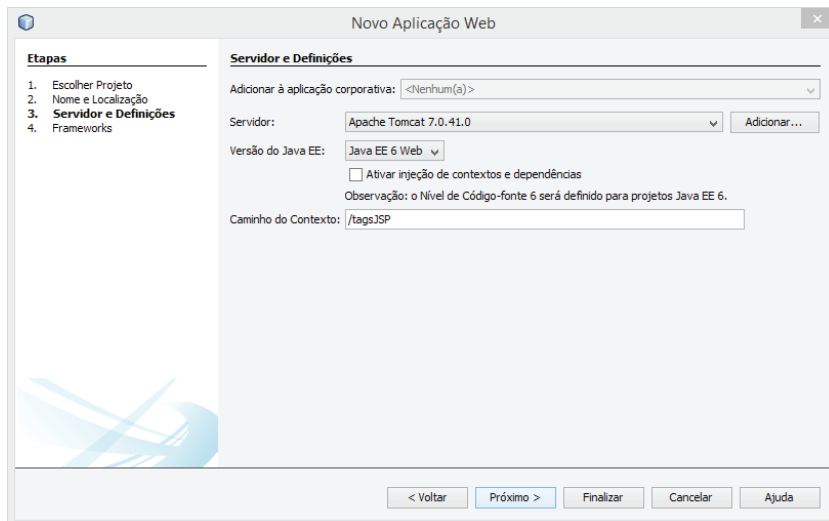


Figura 43 - Escolher servidor Web

PASSO 6

O NetBeans abre uma tela para selecionar *frameworks*, Figura 44, logo após definir o servidor Web, como apresentado na Figura 43. No entanto, não é foco deste livro explorar os diversos *frameworks* para desenvolvimento Web, sendo assim é solicitado ao desenvolvedor clicar diretamente em **Finalizar**.

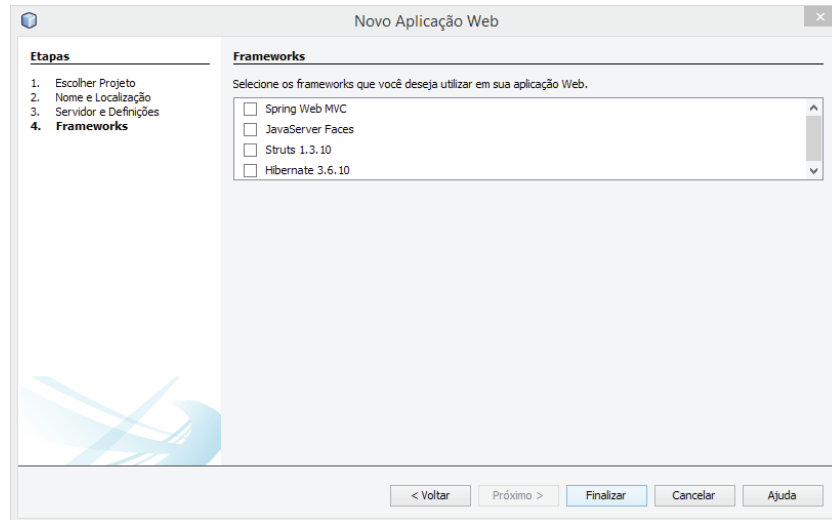


Figura 44 - Definindo *framework*

PASSO 7

O NetBeans criou um projeto de uma aplicação Web chamado **tagsJSP**. Na Figura 45 é apresentado a estrutura da aplicação Web criada pelo NetBeans.

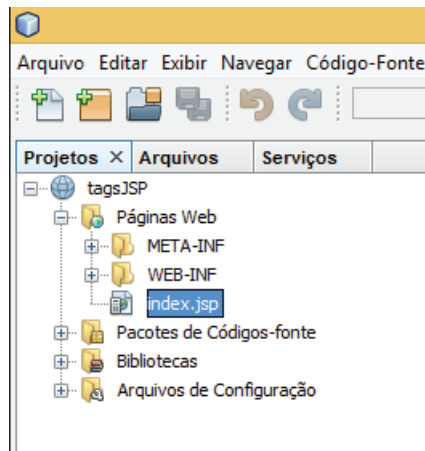


Figura 45 - Estrutura da aplicação Web

PASSO 8

O NetBeans sempre cria uma página JSP chamada **index.jsp** a ser chamada automaticamente ao executar a aplicação. Isso pode ser alterado, no entanto deixe a página do jeito que está e crie uma nova página com o nome **tags.jsp**, clicando em **Arquivo** → **Novo Arquivo...**, como apresentado na Figura 46.

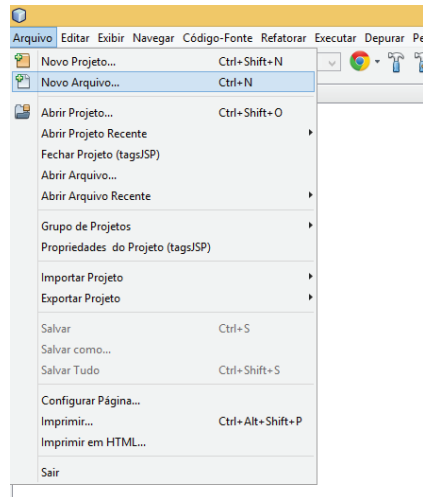


Figura 46 – Novo arquivo

PASSO 9

Ao clicar na opção “Novo Arquivo...” uma tela como apresentado na Figura 47 é aberta. Nesta tela o desenvolvedor deve selecionar qual a **categoria do tipo de arquivo** e o **tipo de arquivo**. Para este exemplo escolha **categoria** igual Web e o **tipo de Arquivos** igual a JSP. Na sequência, clique no botão **Próximo** para continuar.

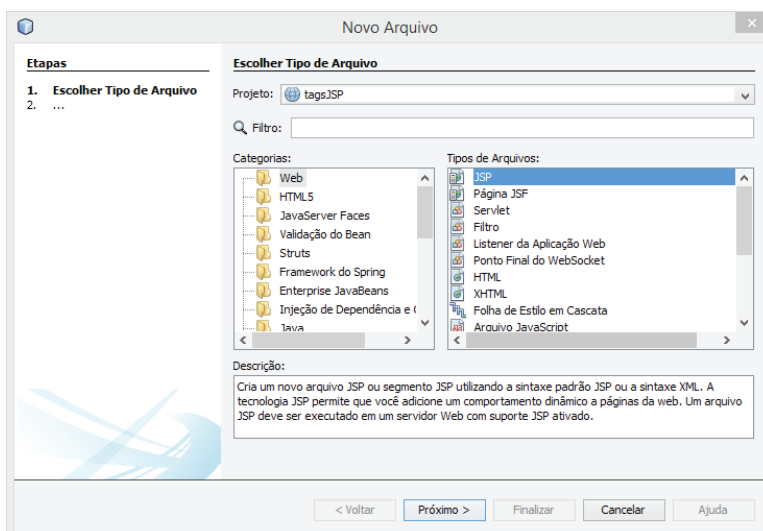


Figura 47 - Criando arquivo JSP

PASSO 10

Na sequência, o NetBeans apresenta uma tela com um campo para inserir o “Nome do Arquivo” e campos para *modificar a localização da pasta*. Para continuar, informe o nome do arquivo igual a **tags**, deixe as configurações padrões e clique em **Finalizar** como apresentado na Figura 48.

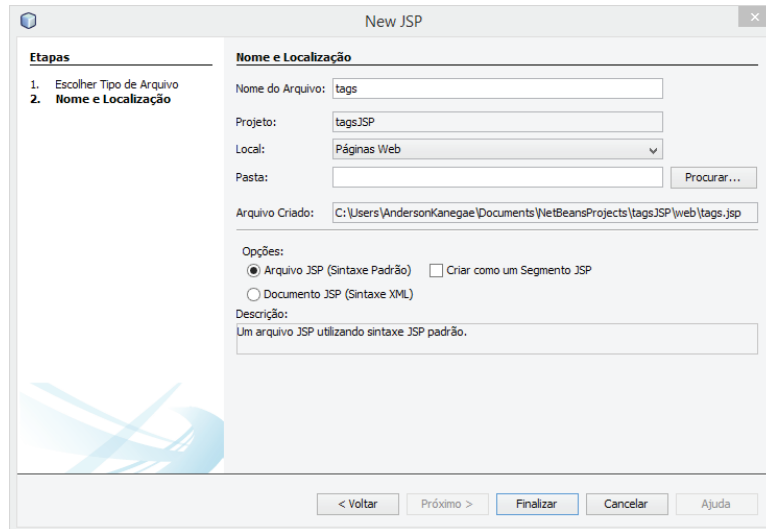


Figura 48 - Nome do arquivo JSP

PASSO 11

Após clicar no botão **Finalizar**, o NetBeans cria um arquivo JSP com nome **tags.jsp**. Abra o arquivo **tags.jsp**, apague todo o conteúdo e digite o código apresentado a seguir:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Exemplo 1 - JSP</title>
    </head>
    <body>
        <h1>Usando JSP</h1>
        <!-- Usando as tags de JSP -->
        <ul>
            <li><p><strong>Expressao</strong></p><p>O valor de
PI eh: <%= Math.PI%></p></li>
            <li><p><strong>Scriptlet</strong></p>
                <p><% out.println("Dados atachados:" + request.get-
QueryString());%></p></li>
```

```

        <li><p><strong>Declaracao</strong></p>
        <p><%! private int contAcesso = 10;%> Acessos para a
pagina desde que o servidor foi reinicializado:<%= ++contAces-
so%></p></li>
        <li><p><strong>Diretiva</strong></p>
        <p><%@ page import = "java.util.*" %>Data corrente:
<%= new Date()%></p></li>
        <li><p><strong>Acao include</strong></p>
        <p><jsp:include page="oi.jsp" flush="true" /></p>
        </li>
    </ul>
</body>
</html>

```

Repare que no **código anterior** existe um tag denominada `<jsp:include...>` que permite inserir uma página JSP dinâmica ou estática dentro de um arquivo JSP.

Por isso a próxima etapa deve ser o desenvolvimento de um novo arquivo JSP com o nome **oi.jsp**. Então, siga as sequências de passos apresentada a seguir.

Página: oi.jsp

PASSO 1

Para criar uma nova página JSP clique em **Arquivo → Novo arquivo...**, selecione a categoria igual Web, tipo de arquivos igual JSP e clique no botão **Próximo**.

Na próxima tela o NetBeans solicita que seja informado o nome do arquivo que deve ser igual a **oi**. Nesta tela o usuário pode modificar a pasta, mas não há necessidade de mudar a localização do arquivo ou pasta. Assim, clique em **Finalizar** e um novo arquivo JSP é criado.

Caso haja alguma dúvida em relação a sequência de passos para criação do JSP, vide os **passos 8, 9 e 10 para criar a página tags.jsp para sanar dúvidas**. A sequência é a mesma, o que muda é o nome do arquivo JSP.

PASSO 2

Após a criação do arquivo JSP, o desenvolvedor deve apagar todo o conteúdo do JSP criado. Sendo assim, digite a seguinte frase apresentada a seguir na nova página **oi.jsp**.

Ola alunos de BSI!

Criando as duas páginas JSP apresentada neste exemplo a primeira aplicação Web está pronta!

Para executar somente o arquivo **tags.jsp**, clique sobre o arquivo com o botão direito e escolha a opção **Executar Arquivo** como apresentado na Figura 49.

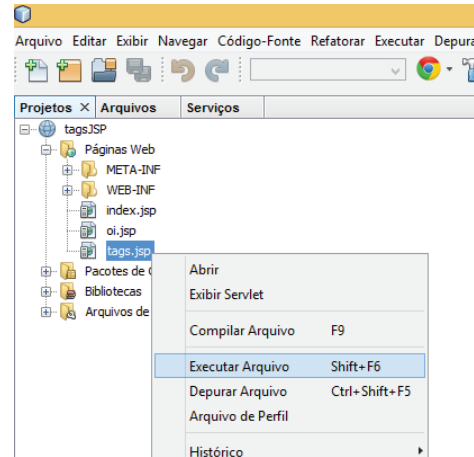


Figura 49 - Executar Arquivo

Outra forma de executar uma aplicação no NetBeans é clicar no botão denominado “Executar Projeto”.

Entretanto, caso clicar no botão “Executar Projeto” vai ser executado o arquivo *default* da aplicação Web. Para este caso, o arquivo **index.jsp**.

Para executar o arquivo **tags.jsp** é preciso digitar o caminho do arquivo manualmente como apresentado na Figura 50. Lembre-se de que a porta do *localhost* pode ser diferente de um usuário para outro.



Figura 50 - Execução da aplicação Web

4.3.2 Exemplo 2 - Fatorial

O segundo exemplo de aplicação Web usando JSP é a criação de um cálculo de fatorial, para isso siga os seguintes passos apresentados a seguir para criação da página JSP.

Página: fatorial.jsp

PASSO 1

Inicialize o NetBeans.

PASSO 2

Crie um “Novo Projeto...”, para isso clique na opção **Arquivo** → **Novo Projeto** como apresentado na Figura 51.

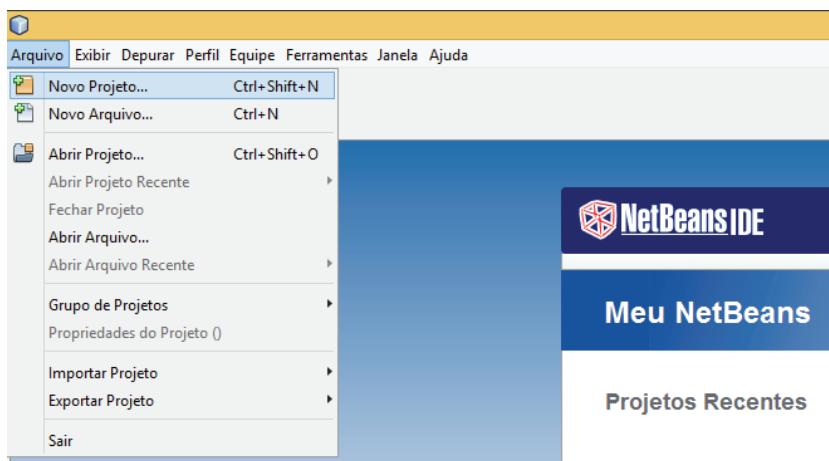


Figura 51 - Novo projeto

PASSO 3

Na tela apresentada na Figura 52 o desenvolvedor deve escolher a **categoria** do projeto igual a Java Web e o **tipo de projeto** igual a Aplicação Web. Na sequência, clique no botão **Próximo**.

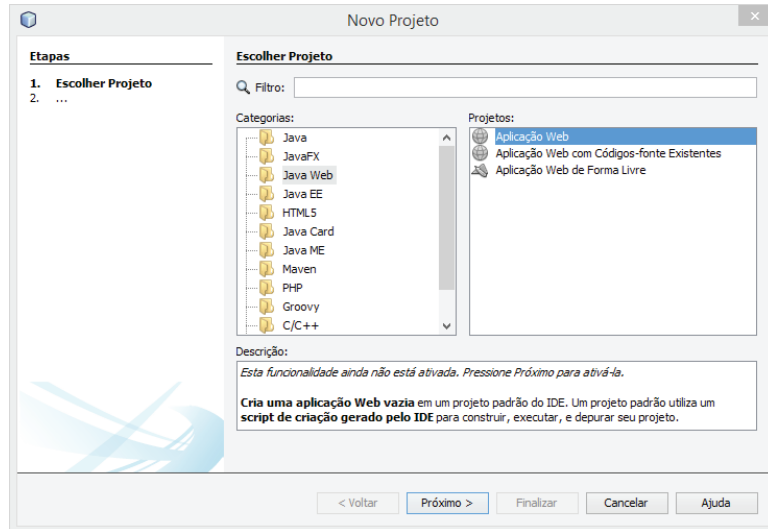


Figura 52 - Criar aplicação Web

PASSO 4

Após definir as características do novo projeto deve ser informado o nome do projeto. Na tela apresentada na Figura 53, deve ser inserido o nome do projeto igual a **fatorialJSP**. Na sequência, clique no botão **Próximo**.

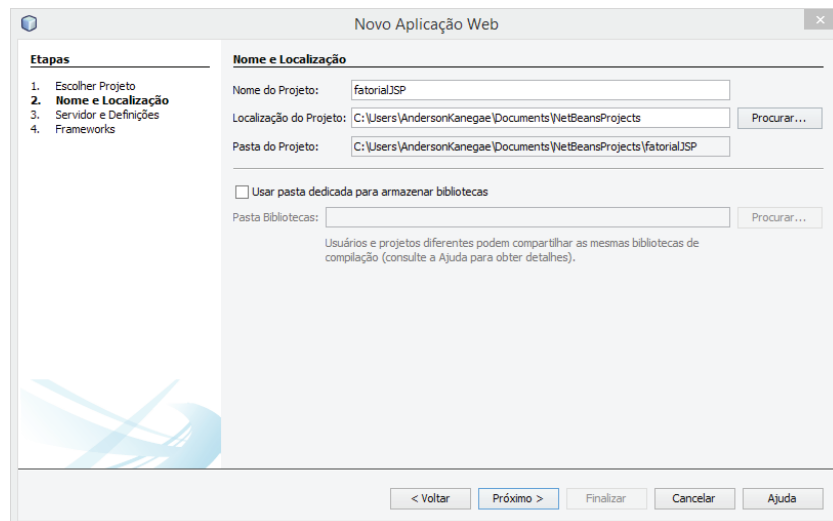


Figura 53 – Arquivo JSP

PASSO 5

O NetBeans direciona a uma tela em que deve ser selecionado o servidor a ser usado na aplicação Web. Escolha o Tomcat na lista e clique no botão **Próximo**, como apresentado na Figura 54.

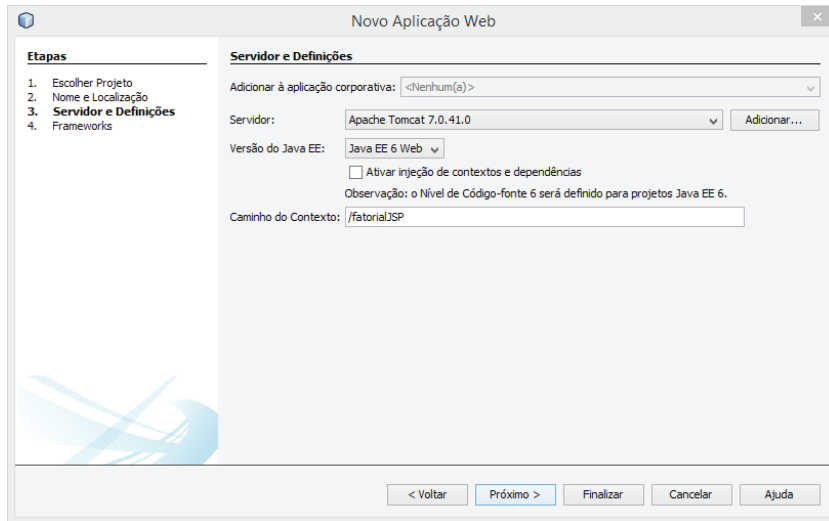


Figura 54 - Sevidor para aplicação Web

PASSO 6

O NetBeans abre uma tela para selecionar *frameworks*, Figura 55, logo após definir o servidor Web, como apresentado na Figura 54. No entanto, não é foco do curso explorar os diversos *frameworks* para desenvolvimento Web, sendo assim é solicitado ao desenvolvedor clicar diretamente em **Finalizar**.

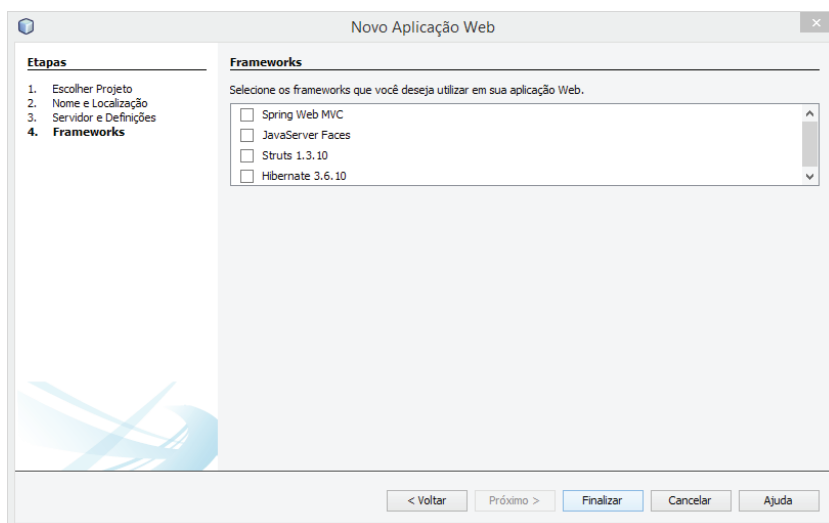


Figura 55 - Definir framework

PASSO 7

O NetBeans criou um projeto de uma aplicação Web chamado **fatorialJSP**. Na Figura 56 é apresentado a estrutura da aplicação Web criada pelo NetBeans.

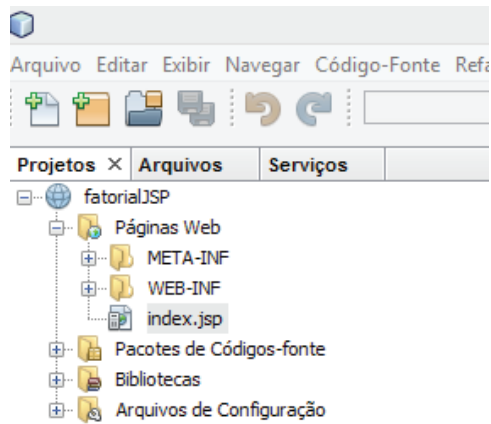


Figura 56 - Estrutura da aplicação Web

PASSO 8:

O NetBeans sempre cria uma página JSP chamada **index.jsp** a ser chamada automaticamente ao executar a aplicação. Isso pode ser alterado, no entanto deixe a página do jeito que está e crie uma nova página com o nome **fatorial.jsp**, clicando em **Arquivo** → **Novo Arquivo...**, como apresentado na Figura 57.

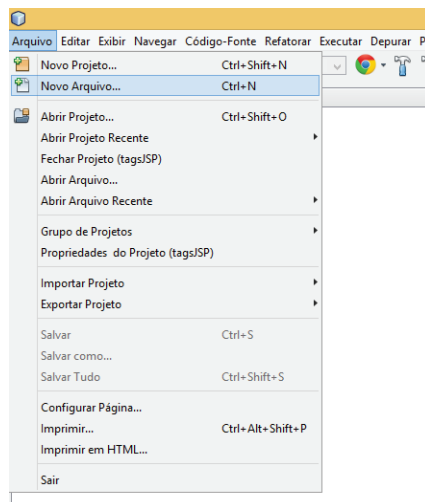


Figura 57 - Criar novo arquivo

PASSO 9

Ao clicar na opção “Novo Arquivo...” uma tela como apresentado na Figura 58 é aberta. Nesta tela o desenvolvedor deve selecionar qual a **categoria do tipo de arquivo** e o **tipo de arquivo**. Para este exemplo escolha **categoria** igual Web e o **tipo de Arquivos** igual a JSP. Na sequência, clique no botão **Próximo** para continuar.

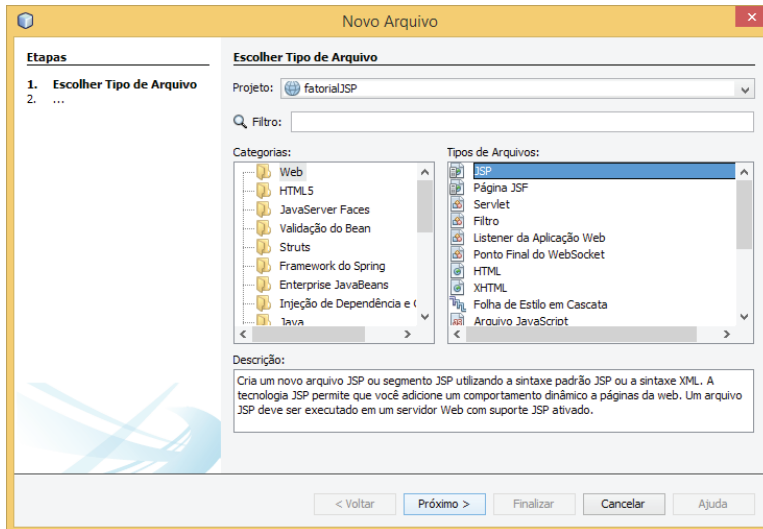


Figura 58 - Arquivo JSP

PASSO 10

Na sequência, o NetBeans apresenta uma tela com um campo para inserir o “Nome do Arquivo” e campos para *modificar a localização da pasta*. Para continuar, informe o nome do arquivo igual a **fatorial**, deixe as configurações padrões e clique em **Finalizar** como apresentado na Figura 59.

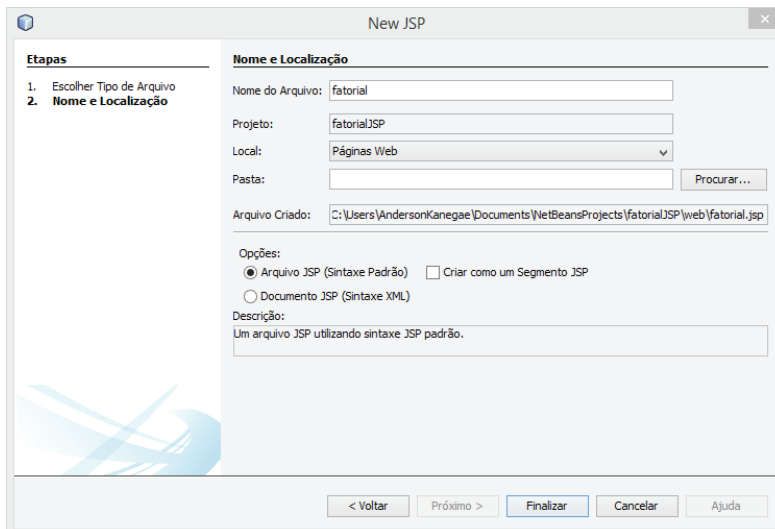


Figura 59 - Nome do arquivo JSP

PASSO 11

Após clicar no botão **Finalizar**, o NetBeans cria um arquivo JSP com nome **fatorial.jsp**. Abra o arquivo **fatorial.jsp**, apague todo o conteúdo e digite o código apresentado a seguir:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Exemplo 2 - JSP / Servidor</title>
  </head>
  <body>
    <h1>Calculo de Fatorial com JSP</h1>
    <%! public long fact(long x) throws IllegalArgumentException
do dominio" );
        if ((x < 0) || (x > 20)) {
            throw new IllegalArgumentException("Fora
do dominio");
        } else if (x == 0) {
            return 1;
        } else {
            return x * fact(x - 1);
        }
    } %>
    <%! public long x = 0;%>
    <% try {
        x = Long.parseLong(request.getParameter("numero"));
        out.print("O fatorial e: " + fact(x));
    } catch (Exception e) {
        if ((x < 0) || (x > 20)) {
            out.print("Nao eh possivel calcular o
fatorial de " + x);
        } else {
            out.print("Erro de Entrada - deveria haver
valor de parametro");
        }
    }
    %>
  </body>
</html>
```

Para finalizar o segundo exemplo **fatorialJSP** é necessário criar um arquivo de extensão **.html**. Sendo assim, siga os seguintes passos para a criação do documento **form.html**.

PASSO 1

Crie um arquivo com extensão **.html** de nome **form.html**. O **documento HTML** gerado deve conter um formulário com um campo de entrada de dados e um botão de envio. Para isso, clique em **Arquivo** → **Novo arquivo**, como apresentado na Figura 60.

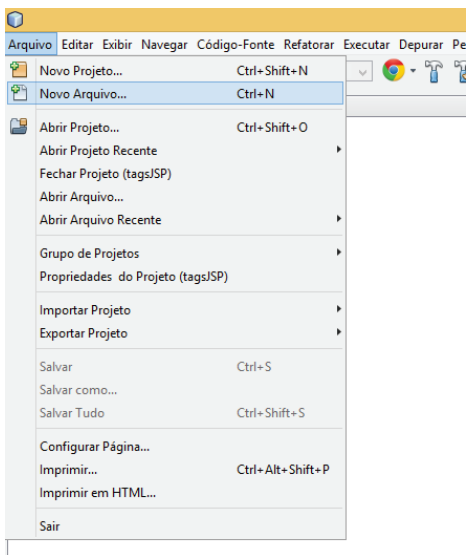


Figura 60 - Novo arquivo

PASSO 2

Ao clicar na opção “Novo Arquivo...” uma tela como apresentado na Figura 61 é aberta. Nesta tela o desenvolvedor deve selecionar qual a **categoria do tipo de arquivo** e o **tipo de arquivo**. Para este exemplo escolha **categoria** igual Web e o **tipo de Arquivos** igual a HTML. Na sequência, clique no botão **Próximo** para continuar.

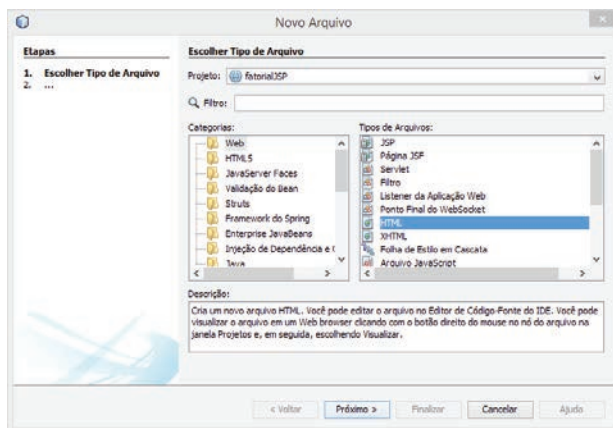


Figura 61 - Arquivo HTML

PASSO 3

Na sequência, o NetBeans apresenta uma tela com um campo para inserir o “Nome do Arquivo” e campos para *modificar a localização da pasta*. Para continuar, informe o nome do arquivo igual a **form**, deixe as configurações padrões e clique em **Finalizar** como apresentado na Figura 62.

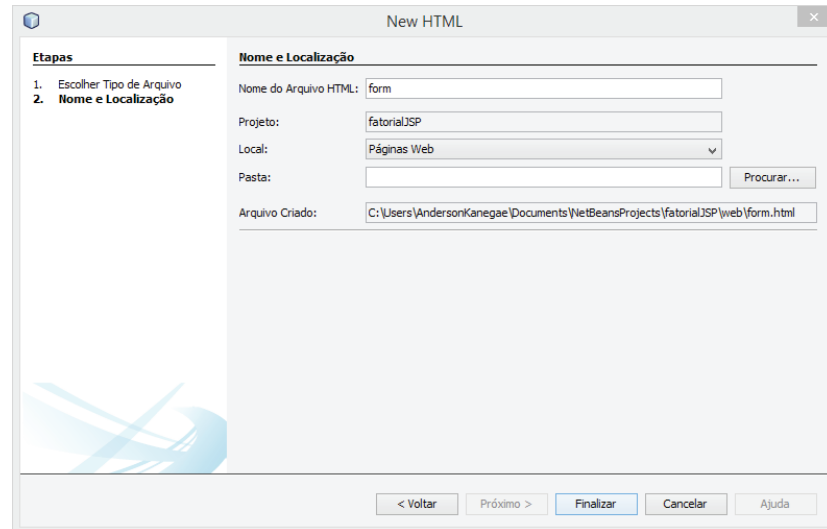


Figura 62 - Criar um arquivo HTML

PASSO 4

Após clicar no botão **Finalizar**, o NetBeans cria um arquivo HTML com nome **form.html**. Abra o arquivo **form.html**, apague todo o conteúdo e digite o código apresentado a seguir:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemplo 2 - JSP / Interface HTML</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width">
  </head>
  <body>
    <h1>Calculo de Fatorial com JSP</h1>
    <p>Calculo do Fatorial</p>
    <form action="fatorial.jsp" method="GET" >
      <input type="text" name="numero" value="0">
      <input type="submit" value="Calcular">
    </form>
  </body>
</html>
```

Pronto, a aplicação Web utilizando JSP está pronta! Altere o arquivo principal da aplicação modificando o *descriptor de implantação*, o **arquivo web.xml**.

Desta vez, vai ser ensinado como criar o **arquivo web.xml** de uma maneira diferente da que foi criada anteriormente, embora o resultado seja o mesmo. Para isso, siga os seguintes passos apresentados a seguir.

Página: web.xml

PASSO 1

Crie um arquivo **web.xml**, clicando em **Arquivo** → **Novo Arquivo...**, como apresentado na Figura 63.

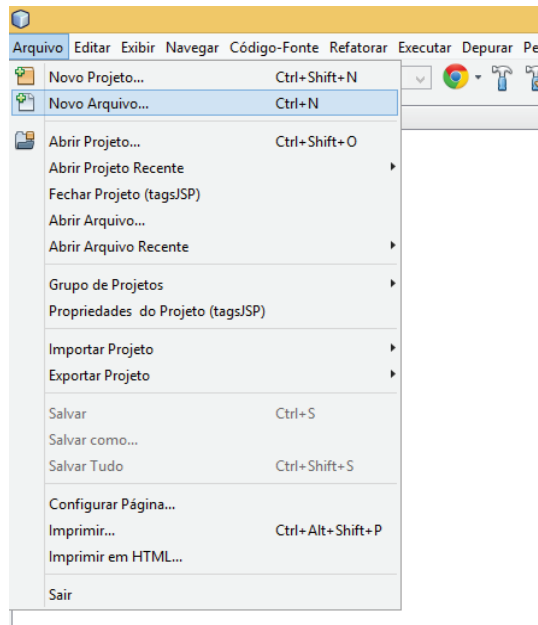


Figura 63 - Criar arquivo

PASSO 2

Escolha a opção Web em categorias, **Descritor de Implantação Padrão (web.xml)** em tipos de arquivos e clique no botão **Próximo**, como apresentado na Figura 64.

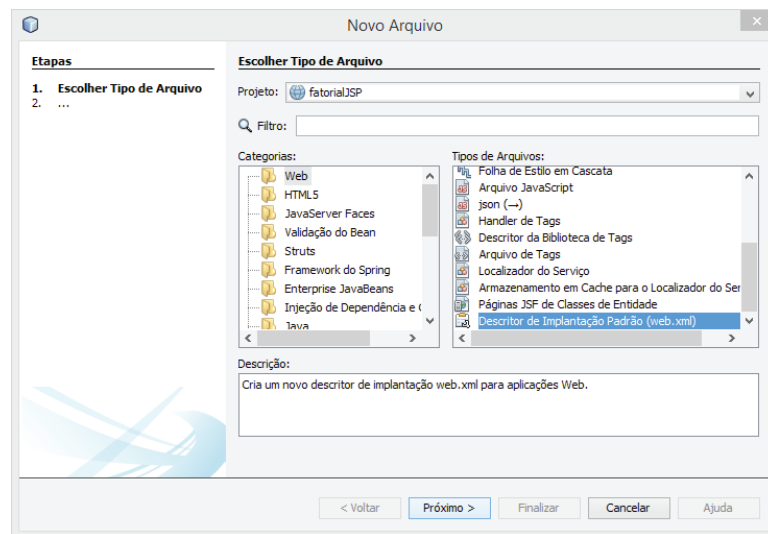


Figura 64 - Selecionando tipo de arquivo

PASSO 3

Após clicar no botão **Próximo**, o NetBeans redireciona para uma tela de confirmação, como apresentado na Figura 65. Assim, confirme os dados e clique em **Finalizar** para criar o **arquivo web.xml**.

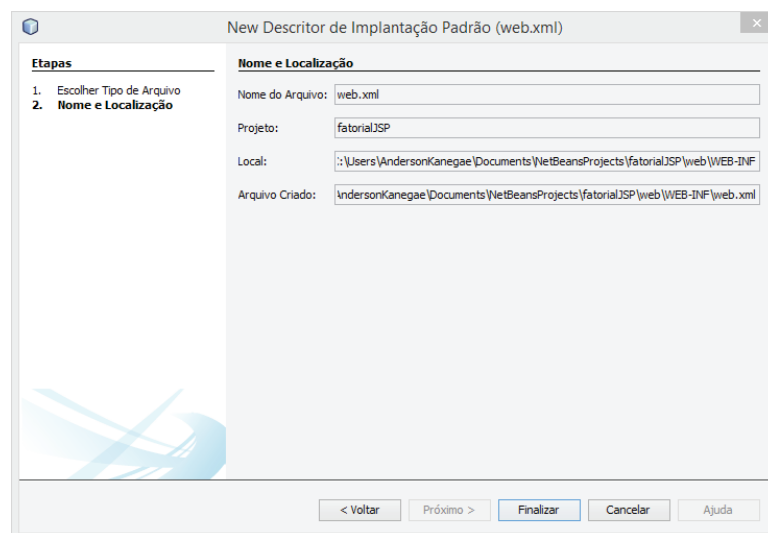
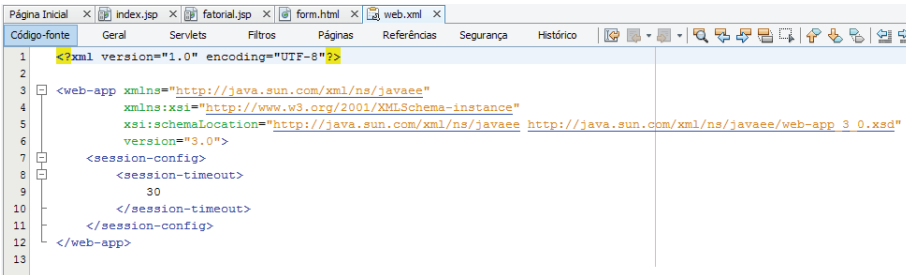


Figura 65 - Confirmação para criação do web.xml

PASSO 4

Após o NetBeans criar o arquivo **web.xml**, abra a pasta **WEB-INF** da estrutura da aplicação Web **fatorialJSP** e localize o arquivo **web.xml**. Ao visualizar a opção código-fonte é aberto uma tela parecida com a Figura 66.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
4          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
6          version="3.0">
7
8   <session-config>
9     <session-timeout>
10      30
11     </session-timeout>
12 </session-config>
13 </web-app>
```

Figura 66 - Arquivo web.xml

PASSO 5

Dê um duplo-clique para editá-lo e a seguinte tela apresentada na Figura 67 vai ser exibida.

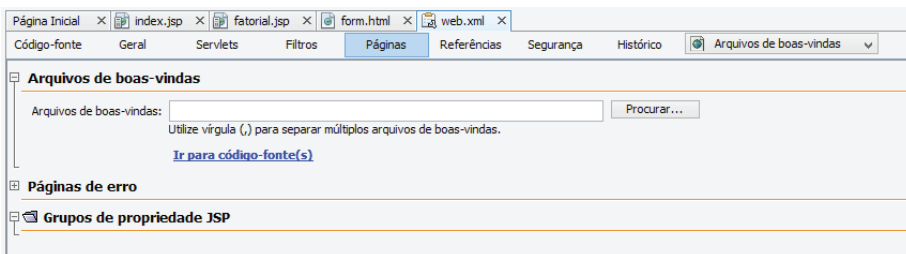


Figura 67 - Editar arquivo web.xml

PASSO 6

No **arquivo web.xml**, clique na aba **Páginas** e no campo “Arquivos de boas-vindas” configure para apontar para o arquivo **form.html**.

Para localizar o arquivo clique no botão **Procurar**, selecione e confirme. Após confirmar, a tela fica igual a Figura 68.

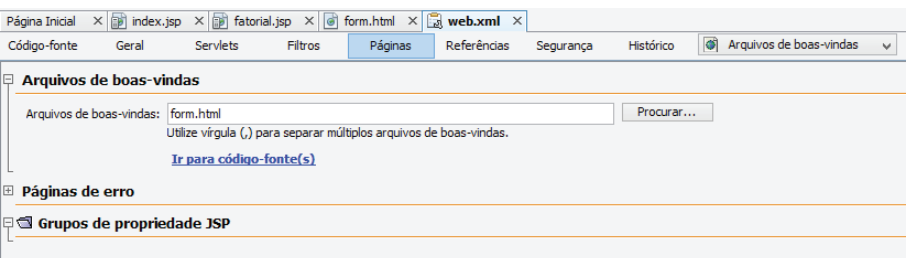


Figura 68 - Editar arquivo web.xml

Após a criação dos arquivos e configuração do web.xml, salve tudo e execute o projeto como já comentado anteriormente. Neste exemplo, clique no botão para “Executar o projeto”.

Ao executar o projeto **fatorialJSP** é possível ver a tela para interagir, apresentado na Figura 69. Já a tela para analisar a resposta decorrente do número a ter o fatorial calculado é apresentado na Figura 70.

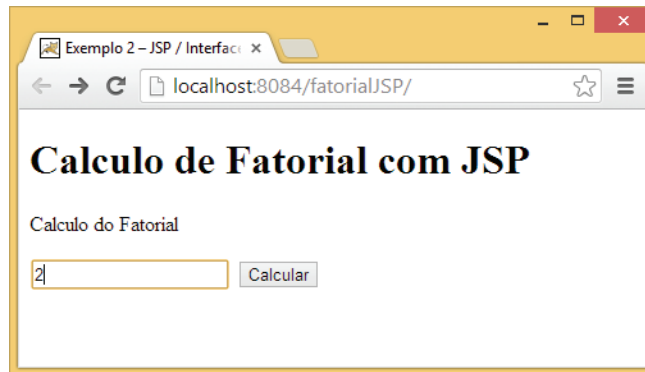


Figura 69 - Página inicial da aplicação Web

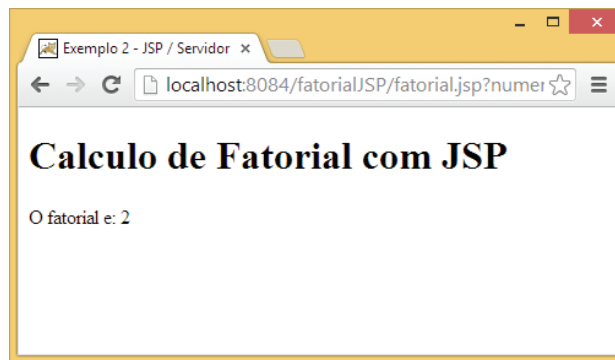


Figura 70 - Cálculo fatorial

4.4 Tratando caracteres especiais

Caracteres especiais precisam ser tratados pelo desenvolvedor. Para evitar problemas relacionadas a acentuação de palavras muito comum na língua portuguesa. O código exemplificado mostra uma página em HTML5 com o encode em UTF-8 e o JSP que recebe o dado submetido pelo usuário com o uso da tag `@page` que especifica o contentType e o pageEncoding. Junto a essa tag é especificado o charset no html e por meio do scriptlet é especificado o tipo de conteúdo no response e no request respectivamente. Observe o código a seguir:

Páginas HTML que envia os dados ao Servlet `acentuacao.jsp`

```
<!DOCTYPE html>
<html>
  <head>
    <title>Palavra</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <form action="acentuacao.jsp" method="POST" >
      <input type="text" name="palavra">
      <input type="submit" value="Calcular">
    </form>
  </body>
</html>
```

Servlet que recebe os dados enviados pela página HTML

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Acentuação</title>
  </head>
  <body>
    <h1>Acentuação</h1>
    <%
      response.setContentType("text/html");
      request.setCharacterEncoding("UTF-8");
      String nome = request.getParameter("palavra");
      out.print(nome);
    %>
  </body>
</html>
```

4.5 Considerações finais

Essa unidade apresentou a tecnologia JSP utilizada no desenvolvimento de aplicações para Web e que permite o processamento de linguagens de script no lado servidor para geração de conteúdo dinâmico.

Foram abordados na unidade o funcionamento do JSP, a estrutura, e os componentes da linguagem.

Para complementar o estudo da tecnologia foi apresentado dois exemplos completos de páginas dinâmicas utilizando a tecnologia JSP desenvolvidos na IDE NetBeans.

4.6 Estudos complementares

Deitel, H. M.; Deitel, P.J. – Java Como Programar. 6ª. Edição. Editora Pearson- Prentice Hall, 2005.

Gonçalves, E. – Desenvolvendo Aplicações Web com NetBeans IDE 5.5 - Editora Ciência Moderna, 2007.

Fields, D.K.; Kolb, M.A. – Desenvolvendo na Web com JavaServer Pages – Editora Ciência Moderna, 2000.

Serson, R.R. - Certificação Java 5. Brasport Livros e Multimidia Ltda, 2006.

UNIDADE 5

Banco de Dados

Banco de Dados

O Banco de Dados mantém um conjunto de registros dispostos em uma estrutura regular que possibilita organizar, armazenar, modificar e extrair dados gerando informação.

Na Unidade 5 é apresentado o Banco de Dados MySQL para gerenciamento de uma base de dados de uma aplicação exemplo.

Ao criar este livro surgiu a preocupação em alertar o leitor sobre as diversas versões que existem no mercado do mesmo produto ou linguagem. Para os exemplos usado neste livro foi usado a última versão do MySQL no período de elaboração do material.

Entretanto, sabe-se que a indústria de tecnologia nunca para e provavelmente o leitor pode se deparar com versões mais recentes do que a abordada inicialmente no livro. Vale lembrar que o tema abordado tem princípios que não sofrem tantas alterações e de que o processo de instalação do banco pode variar um pouco de uma versão para outra.

5.1 Primeiras Palavras

O objetivo da unidade é entender como conectar a página Web com o Banco de Dados MySQL, através da linguagem Java para Web.

5.2 MySQL

O **MySQL** é um Banco de Dados que utiliza a linguagem SQL (Structured Query Language). As principais características são:

- Portabilidade (suporta praticamente qualquer plataforma atual);
- Compatibilidade (existem drivers ODBC, JDBC e .NET e módulos de interface para diversas linguagens de programação, como Delphi, Java, C/C++, Visual basic, Python, Perl, PHP, ASP e Ruby)
- Excelente desempenho e estabilidade;
- Pouco exigente quanto a recursos de hardware;
- Facilidade de uso;
- É um Software Livre com base na GPL;

- Contempla a utilização de vários Storage Engines como MyISAM, InnoDB, Falcon, BDB, Archive, Federated, CSV, Solid, etc.
- Suporta controle transacional;
- Suporta Triggers;
- Suporta Cursors (Non-Scrollable e Non-Updatable);
- Suporta Stored Procedures e Functions;
- Replicação facilmente configurável;
- Interfaces gráficas (MySQL Toolkit) de fácil utilização cedidos pela MySQL Inc.

5.3 Instalando o MySQL

Acesse o site <http://dev.mysql.com/downloads/windows/installer/> e clique na opção `mysql-installer-community-5.6.14.0.msi`, conforme apresentado na Figura 71.

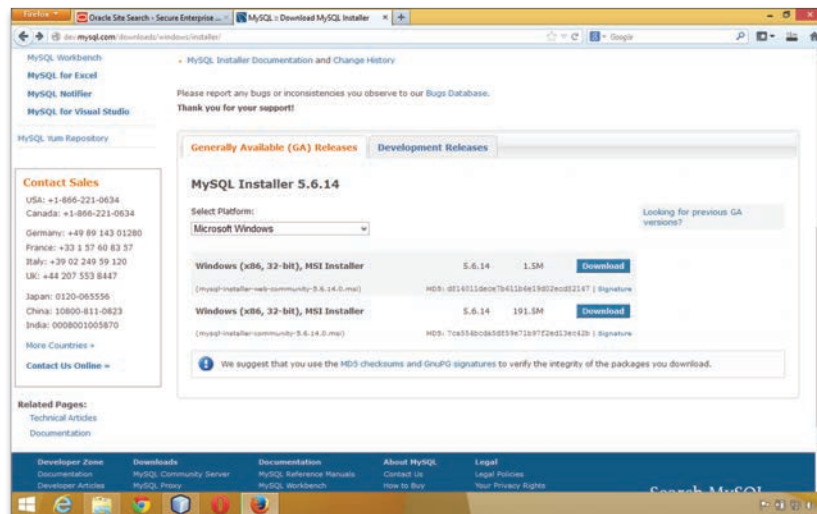


Figura 71 - Página para download do MySQL v 5.6.14

Ao término do download, dê um duplo clique sobre o arquivo. O instalador iniciará a instalação, conforme a Figura 72.

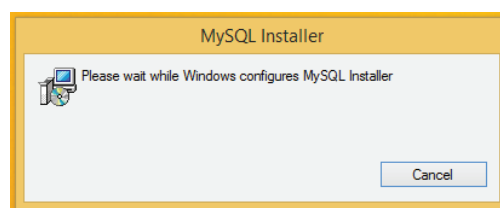


Figura 72 – Instalador MySQL

Após o instalador carregar na memória do computador, uma tela com as opções “Install MySQL Products”, “About MySQL” e “Resources” aparecerá. Conforme apresentado na Figura 73.

→ **Selecione a opção “Install MySQL Products”.**



Figura 73 - Instalação do MySQL

A opção “Install MySQL Products” abrirá um *wizard* para instalação do MySQL.

O *wizard* se inicia com a tela apresentada na Figura 74 em que pede para o usuário aceitar os termos de licença do produto.

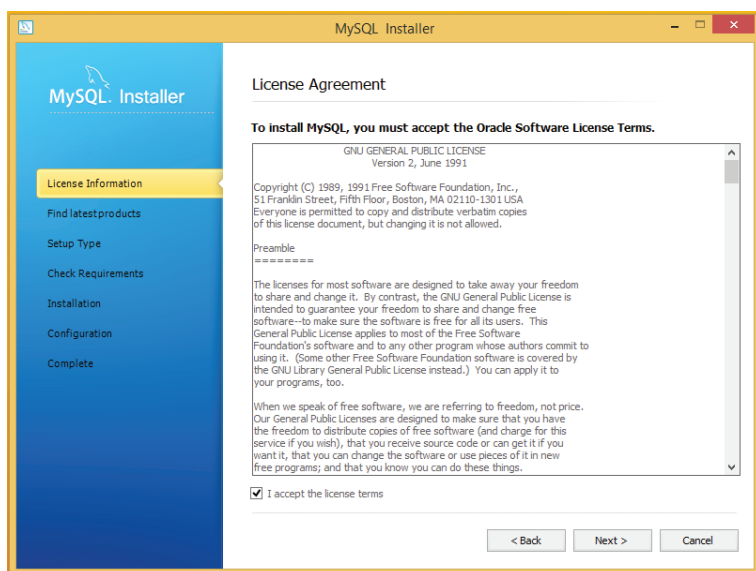


Figura 74 - Confirmação do usuário sobre os termos de licença

Após aceitar os termos de licença, o usuário é conduzido a uma tela para verificar se há uma versão mais atual do banco MySQL que está sendo instalado, veja a Figura 75.

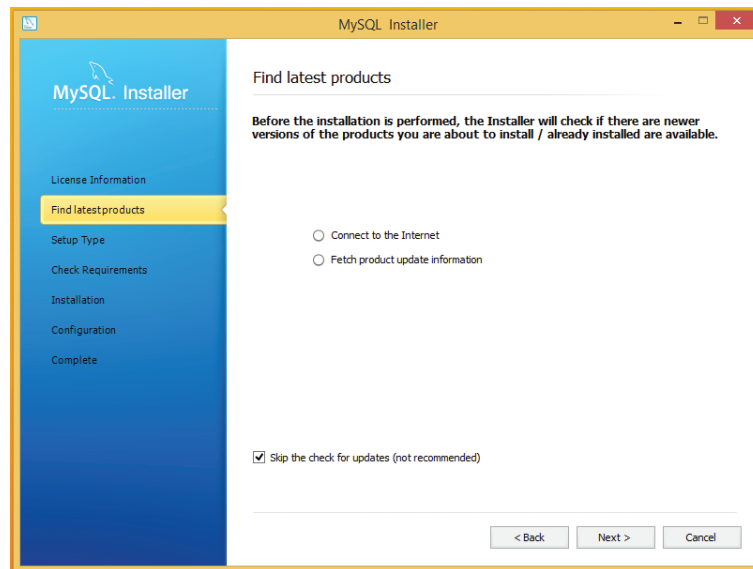


Figura 75 - Procurar por uma versão mais atualizada

Como o download foi feito recente, acredita-se que pode pular esta etapa. Para isso, selecione a opção **“skip the check for update (not recommended)”**.

Ao pular a etapa para verificar a versão recente, o instalador apresenta uma tela em que o usuário deve selecionar qual o tipo de instalação melhor adequada a necessidade do usuário como apresentado na Figura 76.

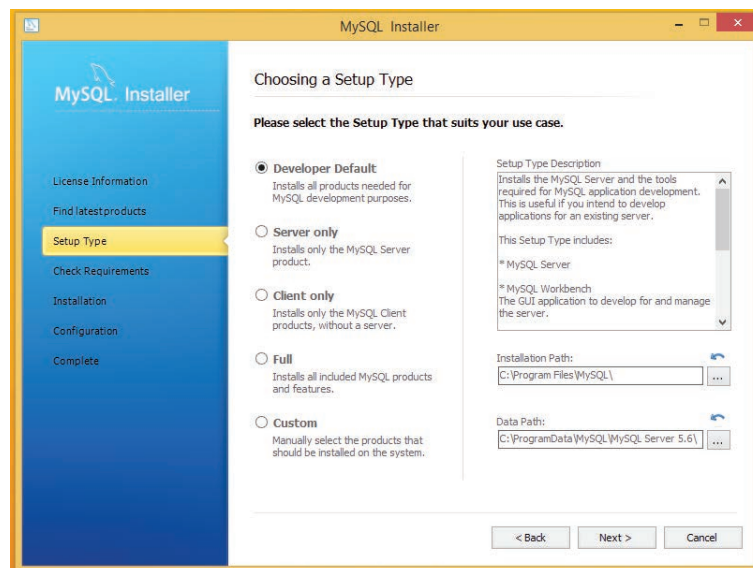


Figura 76 - Selecionar tipo de instalação

O usuário deve selecionar a opção “**Developer Default**”, como apresentado na Figura 76. Os caminhos padrões para instalação (*Installation Path*) e dados (*Data Path*) não precisam ser modificados.

Após selecionado a opção desejada, clique em **NEXT** para prosseguir o *wizard de instalação*.

No próximo passo o programa de instalação verifica se precisa instalar algum *software* adicional, como apresentado na Figura 77 e Figura 78.

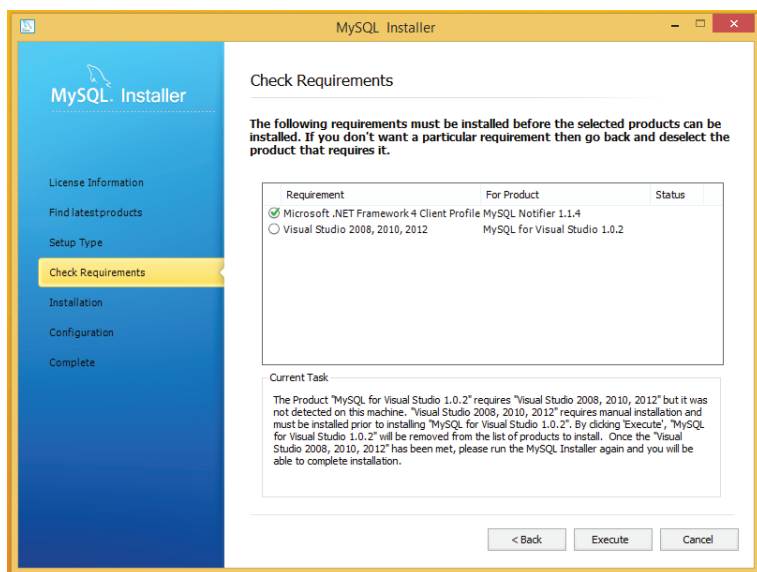


Figura 77 – Verifica requisitos do sistema

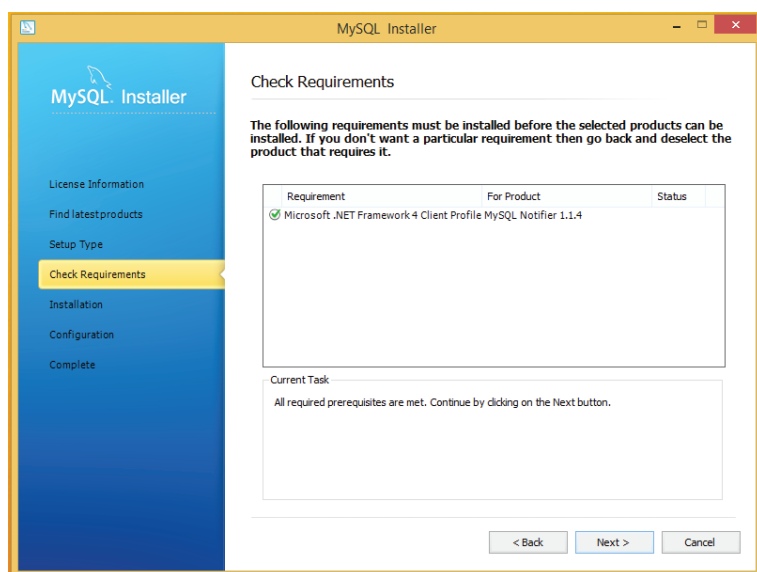


Figura 78 - Verifica requisitos do sistema

Para sair das telas apresentadas nas Figura 77 e Figura 78, clique em “Execute” e “Next” respectivamente. O usuário deve manter as opções padrões apresentada pelo programa de instalação.

Na sequência, o usuário é redirecionado para a tela de progresso de instalação como apresentado na Figura 79 e Figura 80.

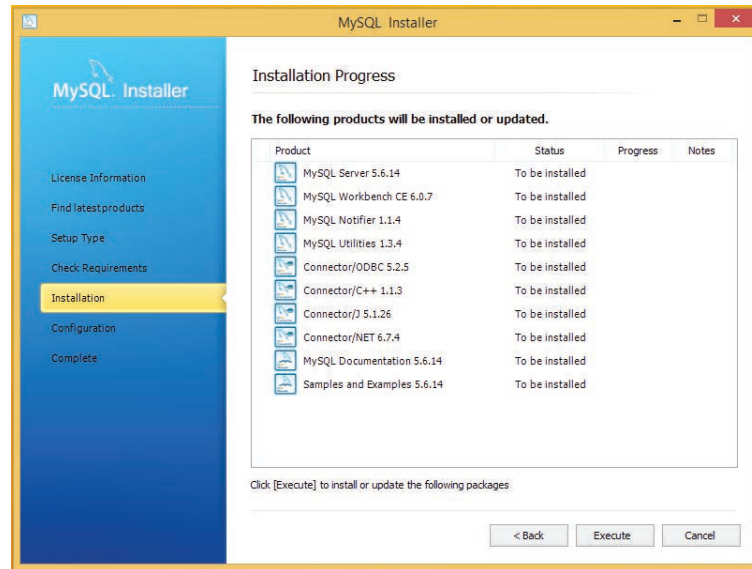


Figura 79 - Progresso de instalação

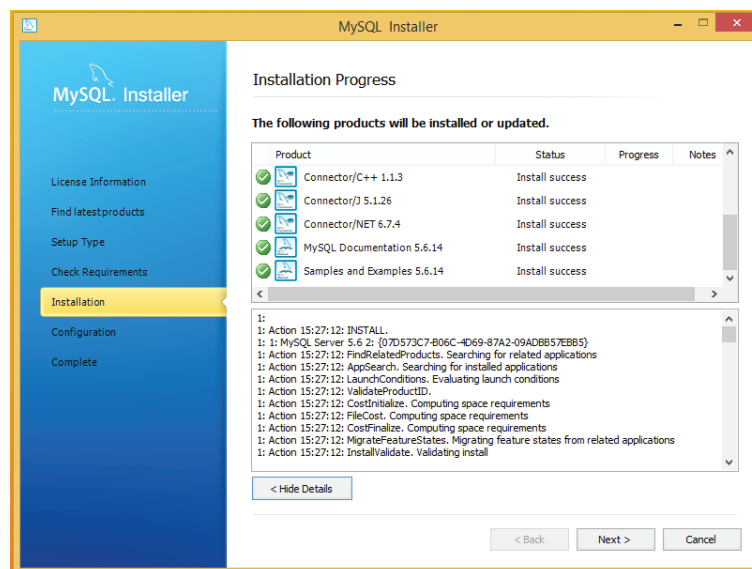


Figura 80 - Progresso de Instalação

O usuário deve clicar na opção “Execute” da Figura 79 e “Next” da Figura 80. Nessas telas são apresentados o progresso de instalação individual de cada componente do pacote dos produtos MySQL.

Ao finalizar o processo de instalação, o usuário é conduzido a fazer a configuração do Banco de Dados.

Inicialmente, os produtos são configurados de forma automática, sem intervenção do usuário como apresentado na Figura 81.

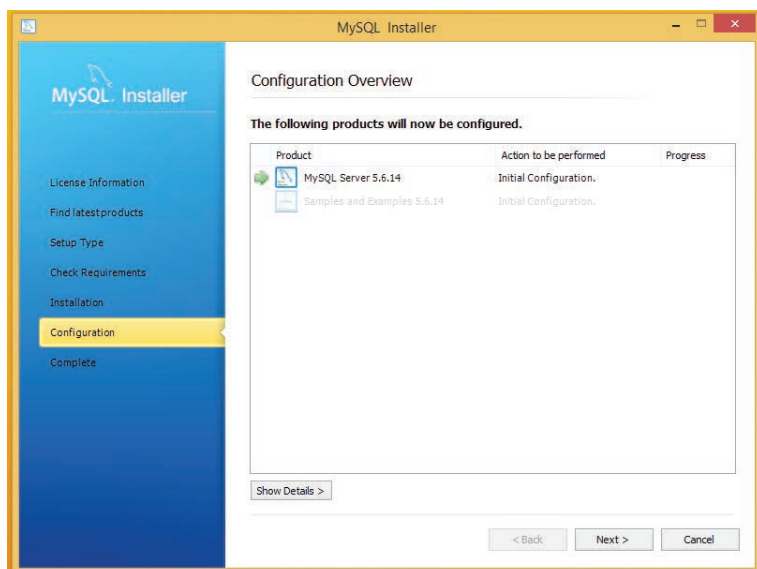


Figura 81 - Configuração de produtos

Quando o usuário clica em **NEXT** para continuar a configuração do MySQL é solicitado ao usuário configurar o tipo de servidor, a porta de conexão do Banco de Dados, a senha do usuário administrador, adição de novos usuários (se for necessário) e configurar detalhes do serviço no Windows. Verifique as telas apresentadas nas Figura 82, Figura 83 e Figura 84.

Na tela inicial das configurações a serem alteradas pelo usuário apresentada na Figura 82, se quiser, não precisa alterar as configurações padrões proposta pelo *software* de instalação.

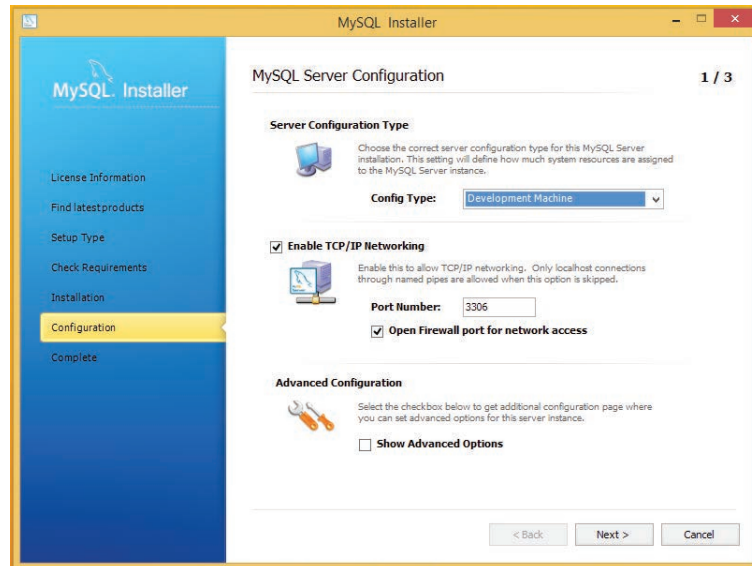


Figura 82 - Configuração

Na segunda tela apresentada na Figura 83, o usuário precisa obrigatoriamente informar uma senha para o usuário root. **Observação:** Como o ambiente é de aprendizado foi inserido uma senha simples, por isso a mensagem “**password strength: Weak**”.

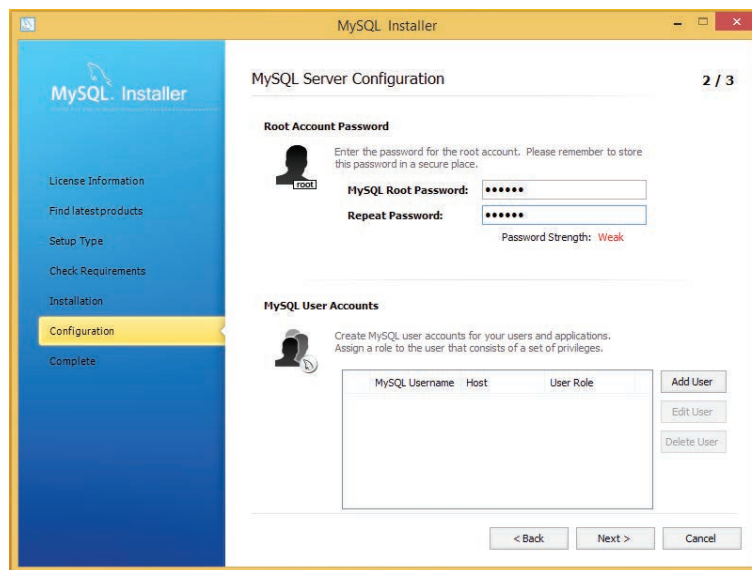


Figura 83 - Configuração

Na tela de configuração, Figura 84, foi deixado as configurações padrões definidas pelo *software* de instalação.

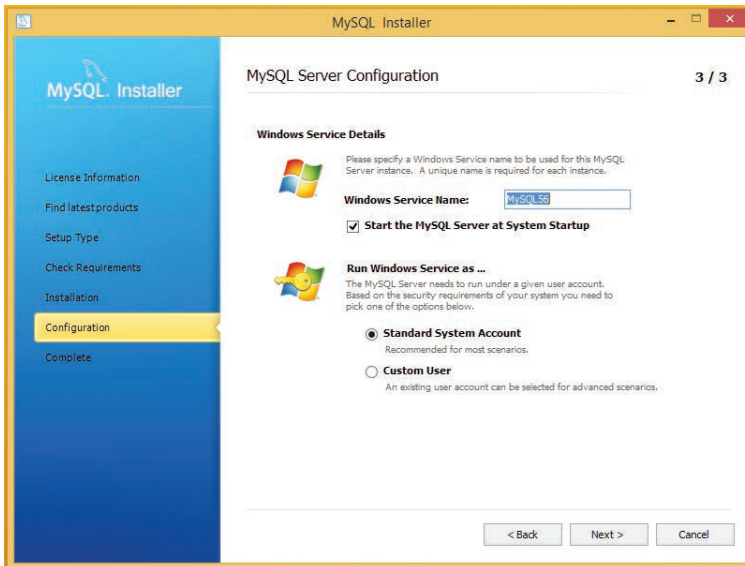


Figura 84 - Configuração

Após definição dos parâmetros, deve-se clicar na opção **NEXT**. Por fim, o *software* de instalação finalizará o processo de configuração MySQL como apresentado na Figura 85.

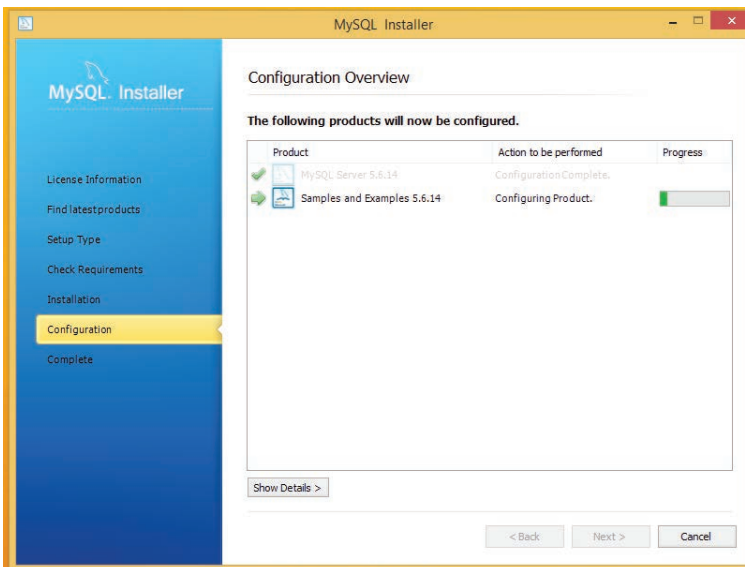


Figura 85 - Finalizar configuração

Ao terminar o processo de configuração do MySQL, o banco é ativo como um serviço no Windows e pode ser acessado na barra de tarefas como apresentado na Figura 86.

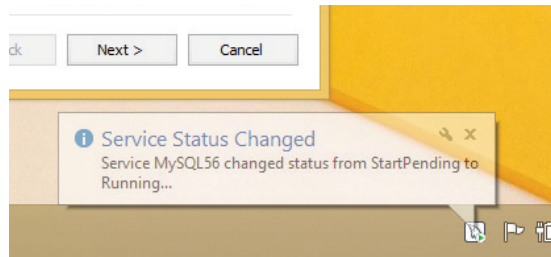


Figura 86 - Serviço no windows

5.3.1 Aplicação com Banco de Dados usando JSP

A aplicação a ser desenvolvida neste exemplo é um sistema de gerenciamento de notícias. Ou seja, a aplicação permite **inserir**, **alterar** e **excluir** uma determinada notícia.

Ao inicializar o NetBeans deve ser visualizado uma tela igual a apresentada na Figura 87.

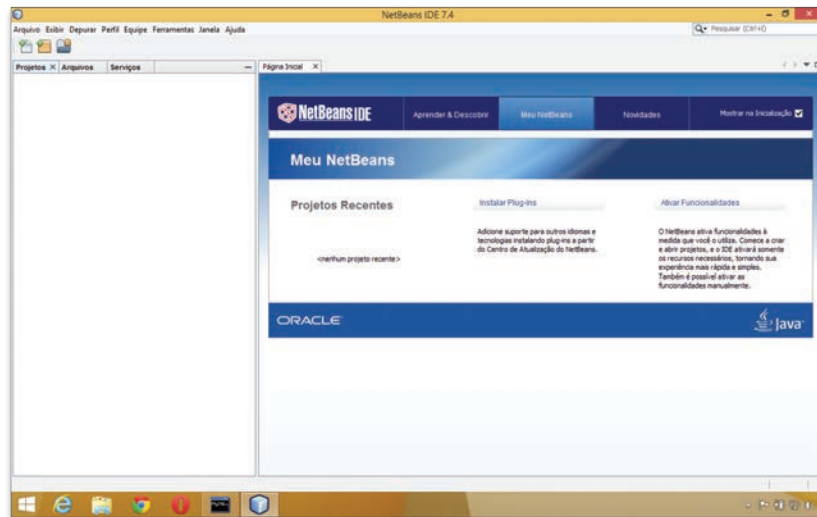


Figura 87 - Tela inicial do NetBeans

Toda aplicação é chamada de projeto. Então, para criar um projeto clique no menu **Arquivo** → **Novo Projeto** como apresentado na Figura 88.

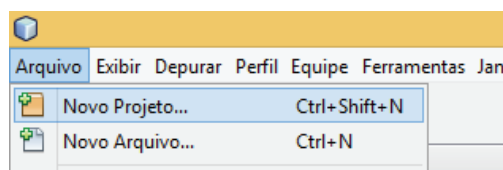


Figura 88 - Criação de projeto

Ao clicar na opção “**Novo Projeto**” surge uma tela que permite seleccionar qual o tipo de projeto e linguagem de programação, como apresentado na Figura 89.

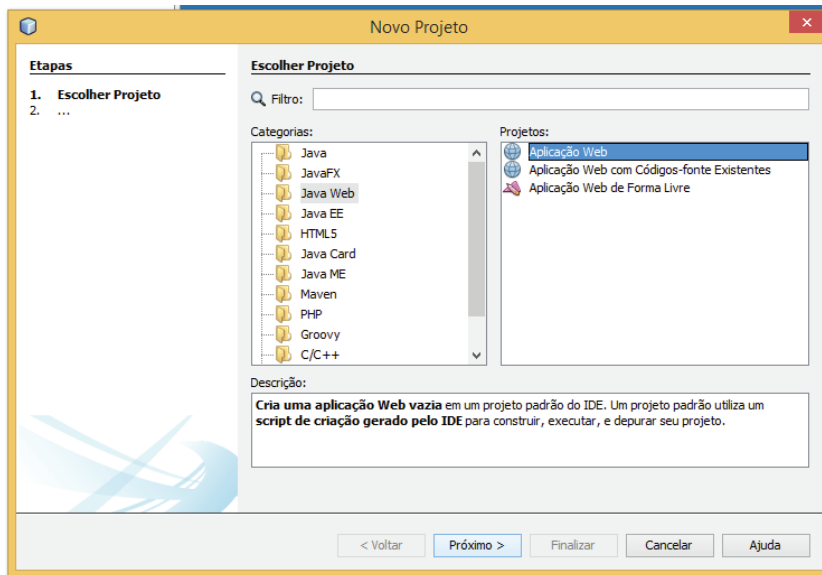


Figura 89 - Seleção do tipo de projeto

A aplicação a ser desenvolvida é para a Web. Sendo assim, selecione a opção **Java Web** na coluna da esquerda e a opção **Aplicação Web** na coluna da direita. Na sequência, clique no botão **Próximo** e o desenvolvedor é guiado a uma tela para informar o nome do projeto e o local onde a aplicação deve ser armazenada, veja a Figura 90.

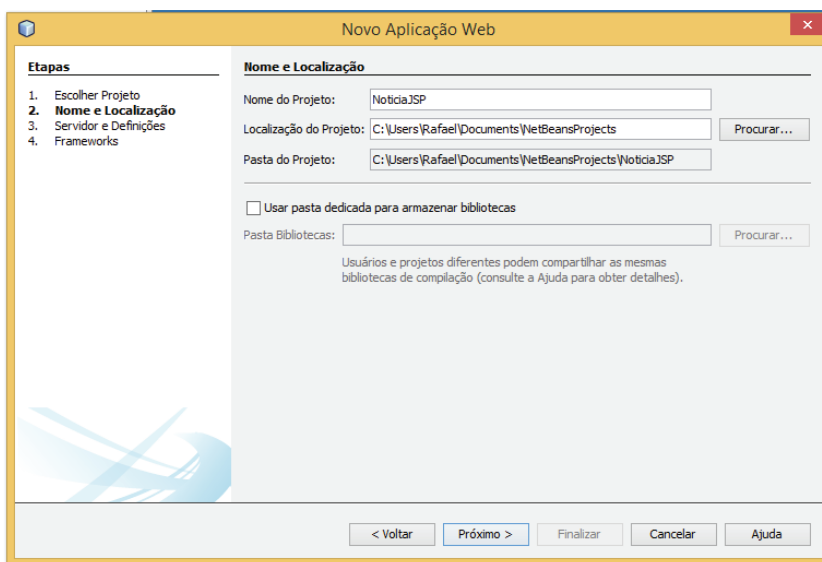


Figura 90 - Configurando criação do projeto Web

Por se tratar de um exemplo, no campo nome do projeto é atribuído o valor **NoticiasJSP**. E ao clicar no botão **Próximo**, a tela para escolher o servidor Web é apresentada, veja a Figura 91.

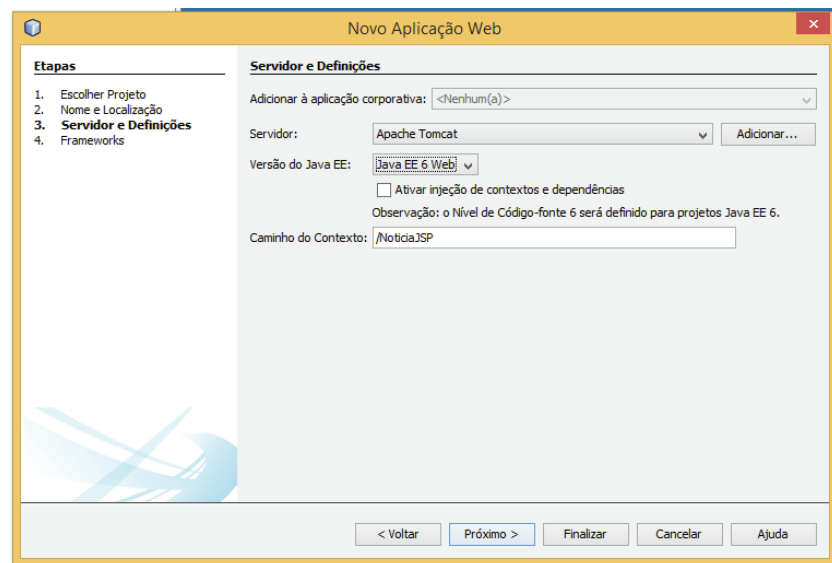


Figura 91 - Definição de servidor

O padrão é o servidor **Glassfish**, mas nos exemplos deve ser utilizado o servidor Tomcat. Então, escolha o **Tomcat** na lista e clique no botão **Próximo**. A tela seguinte, Figura 92, apresenta os *frameworks* disponíveis.

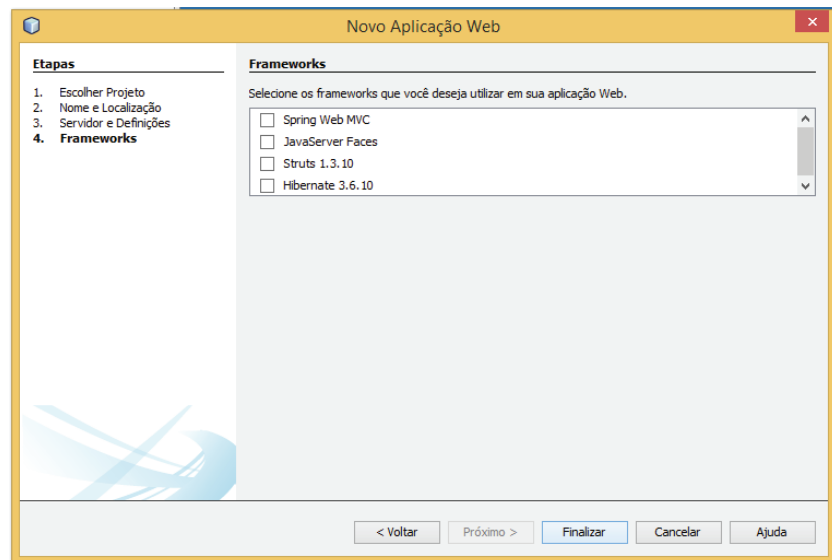


Figura 92 - Framework disponíveis

No curso de Desenvolvimento de Software para Web não vai ser utilizado nenhum *framework*. A ideia deste curso é focar nos conceitos primordiais para a Web considerando que se trata do primeiro contato do aluno com essa tecno-

logia. Em disciplinas posteriores a esta, o conteúdo vai ser mais aprofundado e *frameworks* podem ser abordados.

Então, para continuar clique no botão **Finalizar** e aguarde o NetBeans criar a estrutura do projeto **NoticiasJSP**. Ao término da criação da estrutura, verifique no lado direito da IDE o “*esqueleto*” da aplicação, como apresentado na Figura 93.

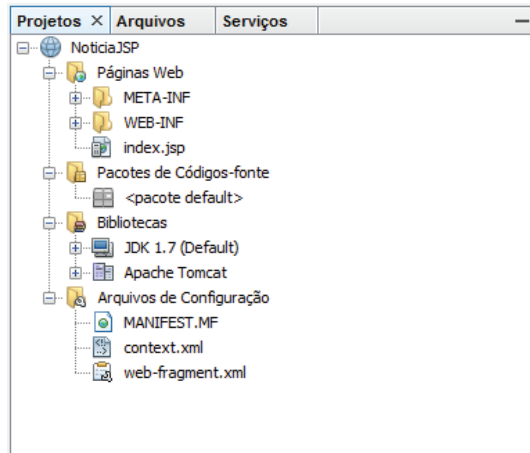


Figura 93 - Estrutura do projeto

O NetBeans sempre cria uma **página JSP** denominada **index.jsp**, executada automaticamente quando a aplicação é iniciada.

Após a criação da estrutura do projeto, precisa ser adicionado a biblioteca do MySQL para comunicação com o Banco de Dados. Para isso, clique com o botão direito sobre a pasta **Bibliotecas** do projeto **NoticiasJSP** e escolha a opção **Adicionar biblioteca**, como apresentado na Figura 94.

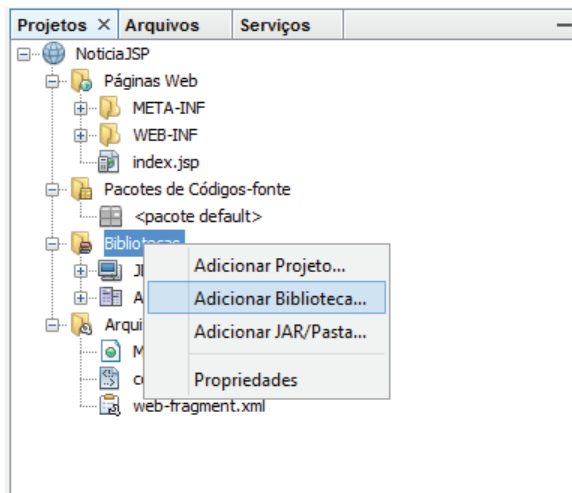


Figura 94 - Adicionar a biblioteca do MySQL

Clicando na opção “Adicionar Biblioteca...” é disponibilizado todas as bibliotecas disponíveis no NetBeans.

O desenvolvedor deve procurar pela biblioteca **Driver JDBC do MySQL** e adicioná-la, como apresentado na Figura 95.

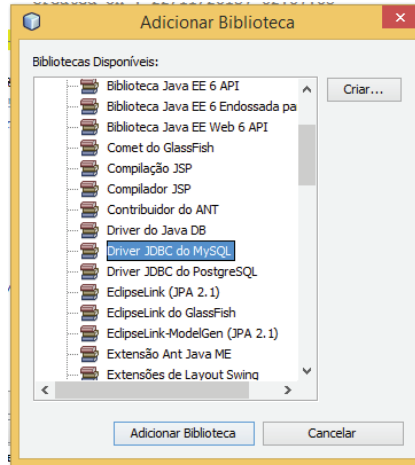


Figura 95 – Bibliotecas

Pronto! A biblioteca do MySQL foi incluída em nosso projeto, como é possível ver na Figura 96 apresentada a seguir.

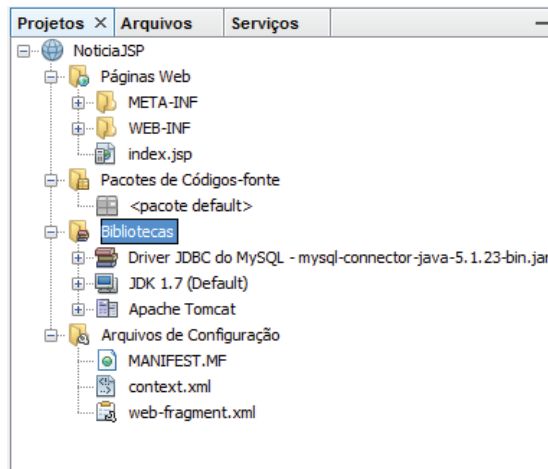


Figura 96 - Biblioteca adicionada

Para a aplicação funcionar precisa ser criado alguns arquivos JSPs. A seguir são apresentados os passos necessários para criação dos arquivos.

Neste momento, não é considerado preocupações com o *layout* e uso de folha de estilo. Isso foi removido nesse momento para facilitar a exposição de ideias sobre o uso do JSP e o Banco de Dados que é o foco desta Unidade.

1º Arquivo JSP – VisualizaNoticia.jsp

Clique com o **botão direito** sob a pasta “*Páginas Web*” que está dentro da estrutura hierárquica de pastas da aplicação Web denominada **NoticiasJSP**. Na sequência, selecione a opção **Novo** → **JSP** como apresentado no Figura 97.

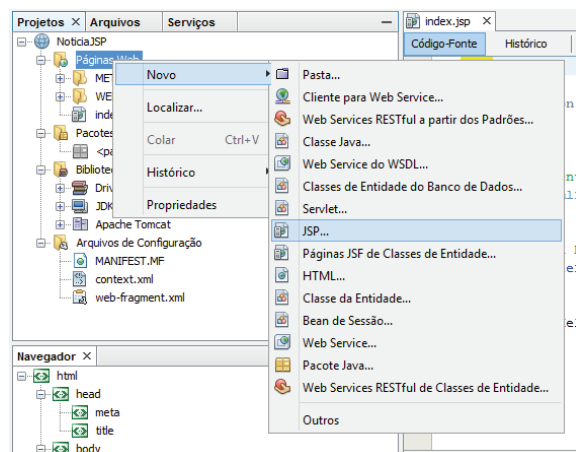


Figura 97 - Criando arquivo JSP

Ao clicar na opção JSP, abre-se uma tela que contém o nome do novo arquivo e sua localização, como apresentado na Figura 98.

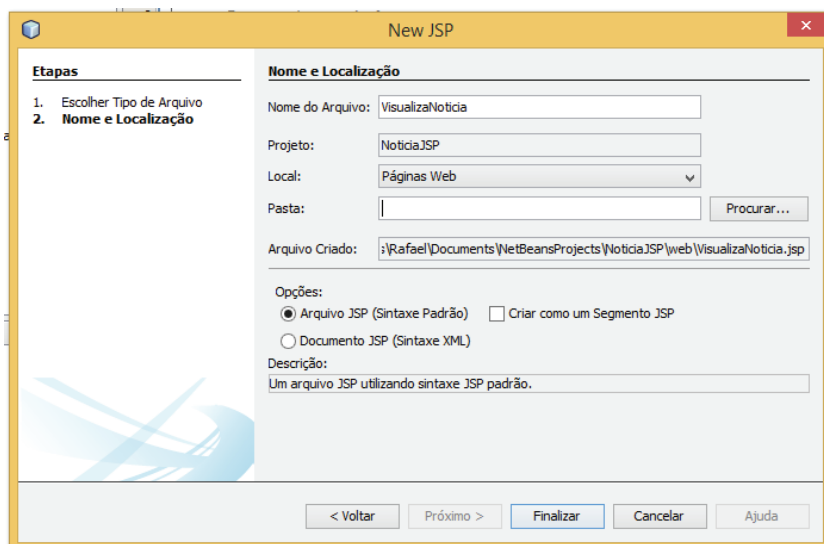


Figura 98 - Criando um novo arquivo JSP

Digite o nome do arquivo como **VisualizaNoticia** (sem a extensão **.jsp**) e clique em **Finalizar**. O arquivo **VisualizaNoticia.jsp** foi criado com sucesso. Apague todo o conteúdo e digite o seguinte código:

```

<%@page contentType="text/html" pageEncoding="iso-8859-1"%>
<%@page import="java.sql.*" %>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="iso-8859-1" />
    <title> Notícias JSP </title>
  </head>
  <body>
    <h1>Visualizar Notícias</h1>
    Notícias cadastradas:<br/><br/>
    <%
      try {
        Class.forName("com.mysql.jdbc.Driver");
        Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"colocar a sua senha");
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM
tbnoticias");
        while (rs.next()) {
          <b>ID:</b> <%= rs.getString("id")%><br>
          <b>Título:</b> <%= rs.getString("titulo")%><br>
          <b>Data:</b> <%= rs.getString("data")%><br>
          <b>Texto:</b> <%= rs.getString("texto")%><br>
          <b>Autor:</b> <%= rs.getString("autor")%><br>
          <br>
        }
        conn.close();
      } catch (Exception e) {
        //ocorreu um erro
      }
    %>
  </body>
</html>

```

2º Arquivo JSP – GerenciaNoticias.jsp

Crie um novo arquivo JSP com o nome **GerenciaNoticias** (caso haja dúvidas sobre a criação de arquivos JSP, reveja como foi criado o primeiro arquivo. Deve ser executado a mesma sequência de passos). Com o novo arquivo criado, apague todo o conteúdo deste arquivo JSP e digite o seguinte código:

```

<%@page contentType="text/html" pageEncoding="iso-8859-1"%>
<%@page import="java.sql.*" %>

```

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="iso-8859-1" />
    <title>Notícias JSP </title>
  </head>
  <body>
    <h1>Gerenciar Notícias</h1>
    Notícias cadastradas:<br/><br/>
    <%
    try {
      Class.forName("com.mysql.jdbc.Driver");
      Connection conn=DriverManager.getConnection("jdbc:mysql://
localhost/bdnoticias", "root", "colocar a sua senha ");
      Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT id, titulo
FROM tbnoticias");
    %>
    <table width="100%" border="1">
    <%
      while (rs.next()) {
    %>
    <tr>
    <td> <%= rs.getString("id")%> </td>
    <td> <%= rs.getString("titulo")%> </td>
    <td><a href="AlterarNoticia.jsp?id=<%= rs.getString("id")%>">
Alterar </a> </td>
    <td> <a href="ExcluirNoticia.jsp?id=<%= rs.getString("id")%>
"> Excluir </a> </td>
    </tr>
    <%
      }
    %>
    </table>
    <br/>
    <a href="InserirNoticia.jsp"> Clique aqui </a> para inse-
rir uma nova notícia.
    <%
      conn.close();
    } catch (Exception e) {
      //ocorreu um erro
    }
    %>
  </body>
</html>

```

3º arquivo JSP – InserirNoticia.jsp

Crie um novo arquivo JSP com o nome **InserirNoticia** (caso haja dúvidas sobre a criação de arquivos JSP, reveja como foi criado o primeiro arquivo. Deve ser executado a mesma sequência de passos). Com o novo arquivo criado, apague todo o conteúdo deste arquivo JSP e digite o seguinte código:

```
<%@page contentType="text/html" pageEncoding="iso-8859-1"%>
<%@page import="java.sql.*" %>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="iso-8859-1" />
    <title>Notícia JSP</title>
  </head>
  <body>
    <%
      if (request.getMethod() != "POST") {
    %>
    <h1>Insere Notícia</h1>
    <br/>
    <form action="InsereNoticia.jsp" method="POST">
    <p>Título: <input type="text" name="txttitulo" /></p>
    <p>Data: <input type="text" name="txtdata" /></p>
    <p>Texto: <textarea name="txttexto"></textarea></p>
    <p>Autor: <input type="text" name="txtautor" /></p>
    <p><input type="submit" value="Insere" name="btninsere" /></p>
    </form>
    <%
      }
      else
      {
        try {
          Class.forName("com.mysql.jdbc.Driver");
          Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"colocar a sua senha");
          Statement stmt = conn.createStatement();
          int rs = stmt.executeUpdate("INSERT INTO
tbnoticias(titulo, data, texto, autor) VALUES(' + re-
quest.getParameter("txttitulo") + ' , ' + re-
quest.getParameter("txtdata") + ' , ' + request.
getParameter("txttexto") + ' , ' + request.
getParameter("txtautor") + ')");
          if (rs == 0) {
    %>
```

```

        Ocorreu um erro na inserção.<br/>
<%
    }
    else
    {
%>
        Notícia inserida com sucesso.<br/>
<%
    }
    conn.close();
} catch (Exception e){
//ocorreu um erro
}
}
%>
</body>
</html>

```

4º arquivo JSP – AlteraNoticia.jsp

Crie um novo arquivo JSP com o nome **AlterarNoticia** (caso haja dúvidas sobre a criação de arquivos JSP, reveja como foi criado o primeiro arquivo. Deve ser executado a mesma sequência de passos). Com o novo arquivo criado, apague todo o conteúdo deste arquivo JSP e digite o seguinte código:

```

<%@page contentType="text/html" pageEncoding="iso-8859-1"%>
<%@page import="java.sql.*" %>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="iso-8859-1" />
        <title>Notícias JSP</title>
    </head>
<body>
    <% if (request.getMethod() != "POST") {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"colocar a sua senha");
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM
tbnoticias where id =" + request.getParameter("id")+";");
            while (rs.next()) {
                %>
                    <h1>Alterar Notícia</h1>
                    <br/>
                    <form action="AlterarNoticia.jsp" method="POST">

```



```

        <p>Título: <input type="text" name="txttitulo"
value="<%= rs.getString("titulo")%>" /></p>
        <p>Data: <input type="text" name="txtdata"
value="<%= rs.getString("data")%>" /></p>
        <p>Texto: <textarea name="txttexto"><%=
rs.getString("texto")%></textarea></p>
        <p>Autor: <input type="text" name="txtautor"
value="<%= rs.getString("autor")%>" /></p>
        <input type="hidden" name="id" value="<%=
rs.getString("id")%>" />
        <p><input type="submit" value="Alterar"
name="btnaltera" /></p>
    </form>
    <%
        }
        conn.close();
    } catch (Exception e) {
        //ocorreu um erro
    }

    } else {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"rafael");

            Statement stmt = conn.createStatement();
            int rs = stmt.executeUpdate("UPDATE tbnoti-
cias SET titulo='\" + request.getParameter("txttitulo") + '\"
, data='\" + request.getParameter("txtdata") + '\" , texto='\"
+ request.getParameter("txttexto") + '\" , autor='\" + re-
quest.getParameter("txtautor") + '\" WHERE id='\" + request.
getParameter("id") + "\";");
            if (rs == 0) {
                <%>
                    Ocorreu um erro na alteração.<br/>
                <%
                    } else {
                <%>
                    Notícia alterada com sucesso.<br/>
                <%
                    }
                    conn.close();
                } catch (Exception e) {
                    // trata erro
                }
            }
        }
    <%>
</body>
</html>

```

5º arquivo JSP – ExcluiNoticia.jsp

Crie um novo arquivo JSP com o nome **ExcluiNoticia** (caso haja dúvidas sobre a criação de arquivos JSP, reveja como foi criado o primeiro arquivo. Deve ser executado a mesma sequência de passos). Com o novo arquivo criado, apague todo o conteúdo deste arquivo JSP e digite o seguinte código:

```
<%@page contentType="text/html" pageEncoding="iso-8859-1"%>
<%@page import="java.sql.*" %>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="iso-8859-1" />
    <title>Notícias JSP</title>
  </head>
  <body>
    <h1>Exclui Notícia </h1>
    <br/>
    <%
      try {
        Class.forName("com.mysql.jdbc.Driver");
        Connection conn=DriverManager.getConnection("jdbc:mysql://localhost/bdnoticias", "root", "colocar a sua senha");
        Statement stmt = conn.createStatement();
        int rs = stmt.executeUpdate("DELETE FROM tbnoticias
WHERE id=" + request.getParameter("id") + ";" );
        if (rs == 0) {
          %>
            Ocorreu um erro na exclusão.<br/>
          <%} else { %>
            Notícia excluída com sucesso.<br/>
          <% } conn.close();
        } catch (Exception e) {
          //tratar erro
        }
      %>
    </body>
  </html>
```

6º arquivo JSP – index.html

Clique com o **botão direito** sob a pasta “*Páginas Web*” que está dentro da estrutura hierárquica de pastas da aplicação Web denominada **NoticiasJSP**. Na sequência, selecione a opção **Novo → HTML**. Na tela que abrir, atribua no campo “*Nome do Arquivo HTML*”, o nome **index** e clique em **Finalizar**. Com o novo arquivo criado, apague todo o conteúdo deste arquivo HTML e digite o seguinte código:

```

<!DOCTYPE html>
<html>
  <head>
    <title> Notícia JSP </title>
    <meta charset="iso-8859-1" />
  </head>
<body>
  <h1>Notícias JSP</h1>
  Escolha a opção abaixo:<br/>
  <ul>
    <li> <a href="VisualizaNoticia.jsp"> Visualizar no-
notícias </a> </li>
    <li><a href="GerenciaNoticias.jsp"> Gerenciar notí-
cias </a> </li>
  </ul>
</body>
</html>

```

Pronto, a primeira aplicação Web está quase pronta! Agora precisa configurar o arquivo web.xml dentro da pasta WEB-INF.

Ao localizar o arquivo **web.xml**, clique duas vezes sob o arquivo para abrir uma tela de configuração do arquivo como apresentado na Figura 99 a seguir.

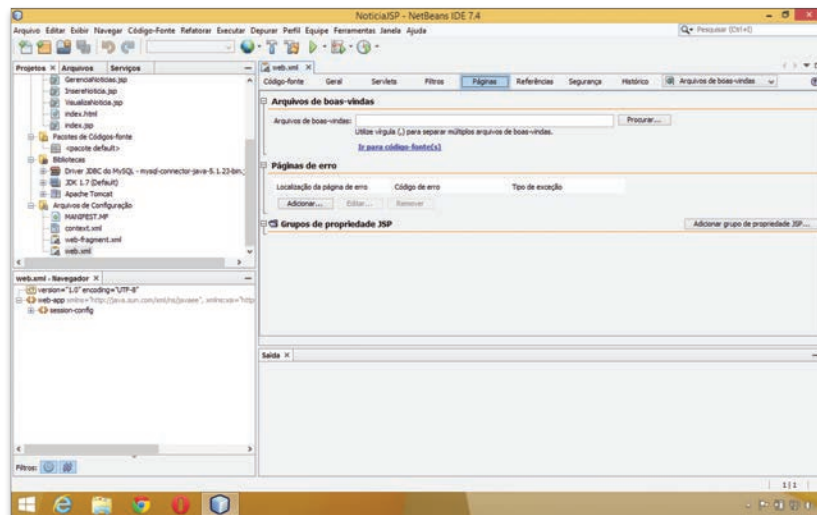


Figura 99 - Configurando o arquivo web.xml

No campo denominado “*Arquivo de boas vindas*” coloque o nome da página **index.html**. Para isso, clique na opção “**Procurar...**”, selecione o arquivo **index.html** e clique em **Selecionar o Arquivo**.

DICA

Caso o arquivo **web.xml** não exista na estrutura da **aplicação Web**, o desenvolvedor pode criar este arquivo.

Para criar o arquivo **web.xml**, basta clicar com o botão direito sob a **pasta WEB-INF** e **selecionar OUTROS**, como apresentado na Figura 100.

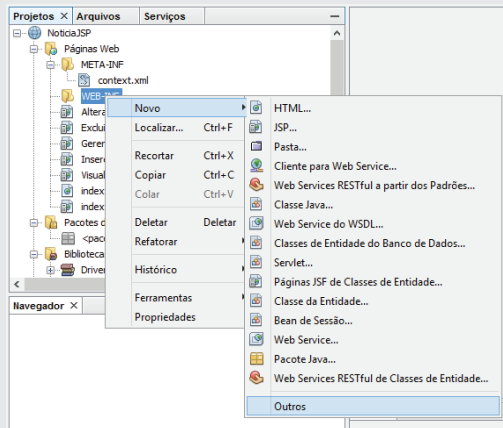


Figura 100 - Arquivo web.xml

Ao clicar em outros, uma tela é aberta solicitando ao desenvolvedor a definição para qual tipo de arquivo deve ser gerado. Assim, selecione a opção *“Descritor de Implantação Padrão (web.xml)”* que está na **categoria Web**, como apresentado na Figura 101.

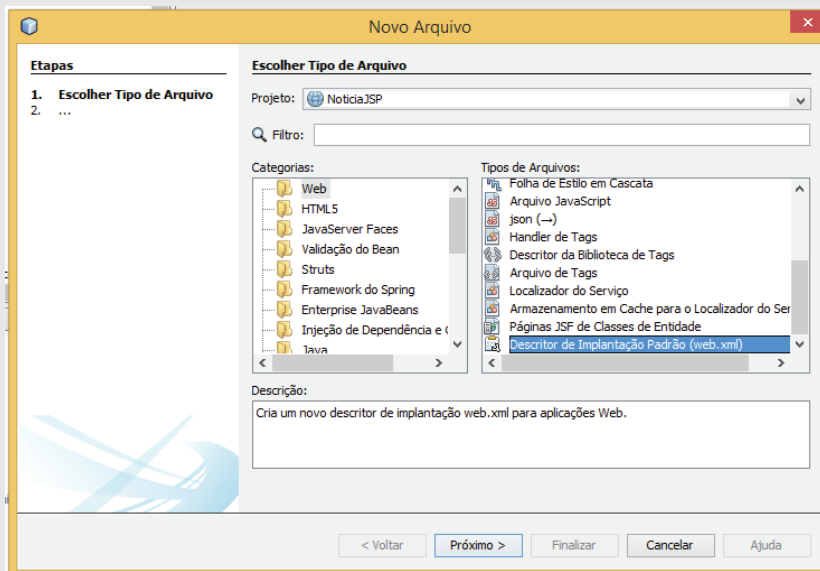
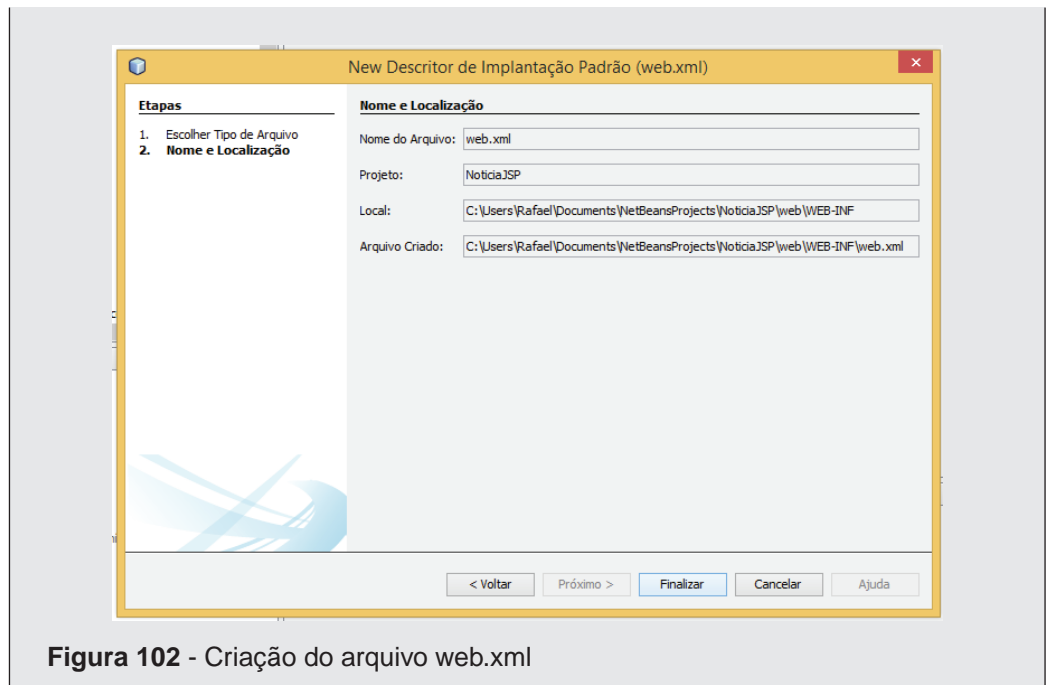


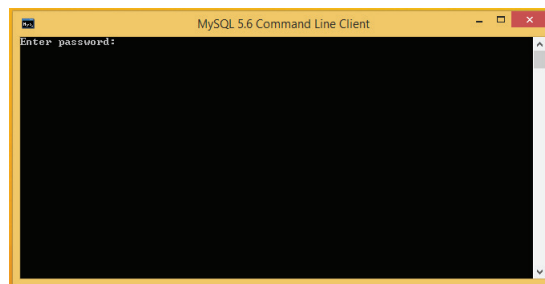
Figura 101 - Criar arquivo web.xml

Após selecionar o tipo de arquivo a ser criado, clique na opção **Próximo** e depois em **Finalizar**, como apresentado na Figura 102.



Para finalizar o exemplo precisa ser criado o Banco de Dados, então procure o aplicativo **MySQL Command Line Client**.

O aplicativo **MySQL Command Line Client** permite acessar o Banco de Dados e inserir comando SQL para criação de tabelas, geração de consultas e etc. Trata-se de um programa executado no **Command Prompt** do Windows como apresentado na Figura 103.



Ao executar o aplicativo do MySQL é solicitado a inserção da senha definida no momento da instalação do Banco de Dados MySQL.

Após digitar a senha, clique na tecla **enter** e a área para inserir comando SQL é liberada como apresentado na Figura 104.

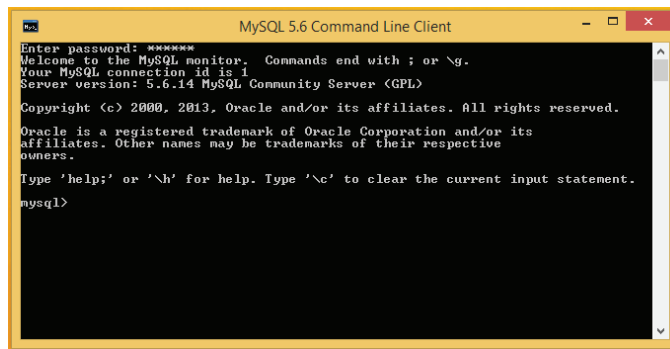


Figura 104 - MySQL Command Line Client

No terminal, igual ao apresentado na Figura 104, digite a a linha de comando “create database bdnoticias;”. Tal linha define a criação de uma base de dados com o nome **bdnoticias**.

Após digitar o comando clique na tecla **enter** e a base de dados é gerada, como apresentado na Figura 105.

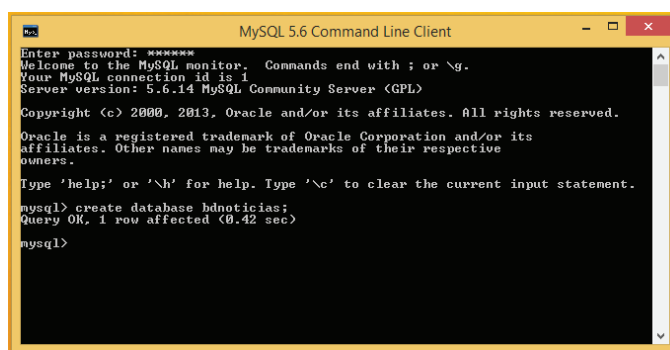


Figura 105 - Criação da base de dados

Com o banco criado é necessário informar qual a base de dados que vai ser utilizada. Assim, use o código “use bdnoticias;” para indicar qual a base vai ser usada a partir desse momento no **MySQL Command Line Client** como apresentado na Figura 106.

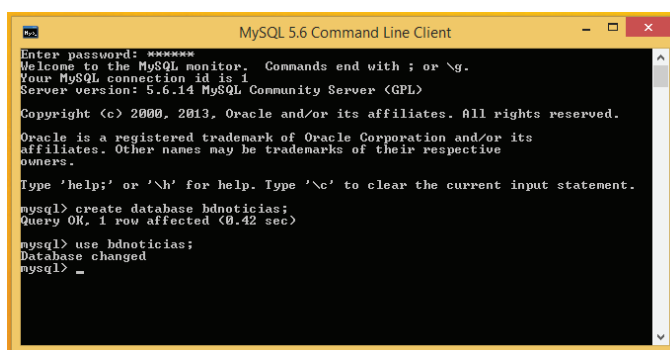


Figura 106 - Acessando o banco de dados criado.

Definido qual o Banco de Dados que vai ser criado as tabelas do sistema de notícias, insira a seguinte linha de comando no terminal:

```
CREATE TABLE `bdnoticias`.`tbnoticias` (  
  `id` int(10) unsigned NOT NULL auto_increment,  
  `data` datetime NOT NULL,  
  `titulo` varchar(100) NOT NULL,  
  `texto` text NOT NULL,  
  `autor` varchar(100) NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

A linha de comando está escrita em **linguagem SQL** e cria uma **tabela de notícias**. A tabela de notícias está com o nome **tbnoticias** e contém um campo identificador auto-incremento, a data de postagem, o título da notícia, o texto e o autor. Após a criação da tabela, digite o comando “quit;” para sair do terminal do **MySQL**.

Voltando ao NetBeans, clique com o *botão direito* sob o projeto “**NoticiasJSP**” e selecione a opção **Executar**.

O projeto é executado, abrindo inicialmente a página **index.html**. Com a página **index.html** aberta no navegador, clique nos **links** para **navegar**, **inserir**, **alterar** e **consultar** os dados.

5.3.2 Aplicação com Banco de Dados usando Servlet

No primeiro exemplo foi usado **JSP** para criação do sistema. Agora neste tópico, vai ser usado **Servlet** para criação da nova **aplicação Web**.

Portanto, crie uma nova **aplicação Web** e coloque o nome do projeto como sendo **NoticiasServlet**.

Nesse novo projeto configure o **servidor TomCat** para rodar a aplicação e configure a **biblioteca do MySQL**.

Tanto a criação da aplicação Web, configuração do servidor e biblioteca já foram apresentadas anteriormente (*vide exemplo com JSP*).

A diferença desse exemplo com o exemplo anterior é que o JSP é substituído por Servlets. A nova aplicação possui alguns Servlets que devem ser criados, para isso siga os passos a seguir.

1º Servlet - VisualizaNoticiasSrv

Clique com o **botão direito** sob a pasta “*Páginas Web*” que está dentro da estrutura hierárquica de pastas da aplicação Web denominada **NoticiasServlet**. Na sequência, selecione a opção **Novo** → **Servlet** como apresentado no Figura 107.

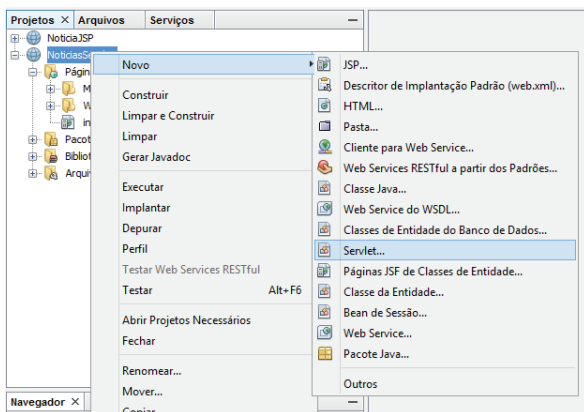


Figura 107 - Criando arquivo JSP

Ao clicar na opção **Servlet...**, abre-se uma tela que contém o nome do novo arquivo e sua localização, como apresentado na Figura 108.

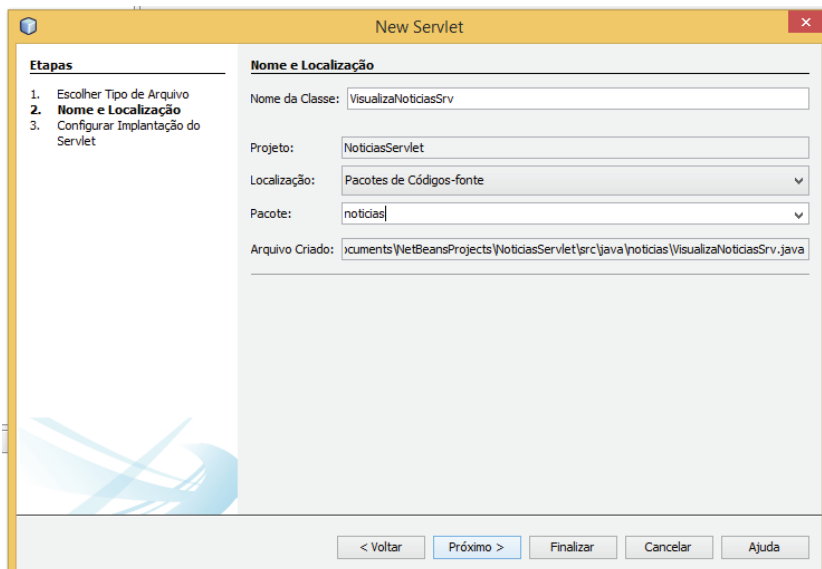


Figura 108 - Criando um novo arquivo JSP

Digite o nome da classe Java como **VisualizaNoticiasSrv** (sem a extensão **.java**), defina o pacote como **noticias** e clique em **Próximo**. A tela apresentada na Figura 109, contém as informações de mapeamento do servlet que está sendo criado.

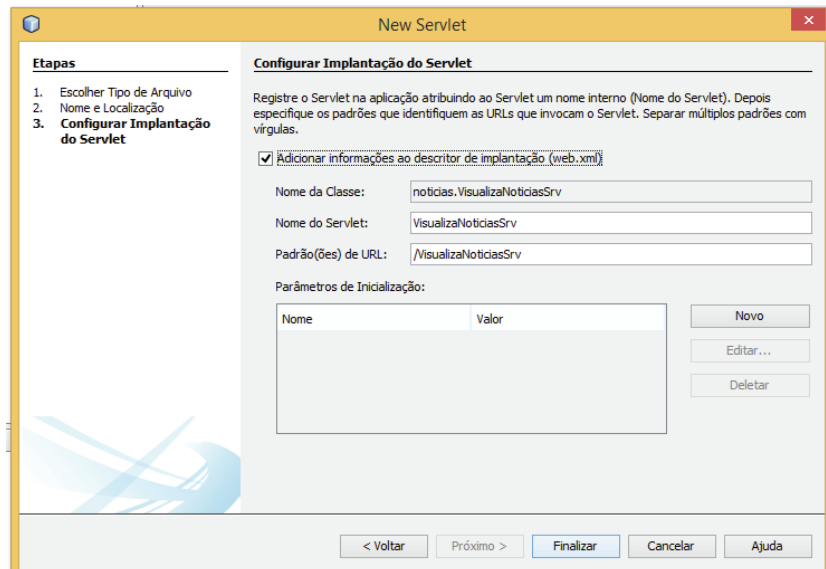


Figura 109 - Criação de Servlet

O NetBeans configura o arquivo **web.xml** automaticamente, assim não precisa se preocupar com o mapeamento. Finalize a criação do arquivo clicando na opção **Finalizar**.

Apague o conteúdo do arquivo **VisualizaNoticiasSrv.java** e digite o seguinte código:

```
package noticias;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class VisualizaNoticiasSrv extends HttpServlet {

    protected void processRequest (HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOEx-
    ception {
        response.setContentType ("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Not&iacute;cias Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Visualizar Not&iacute;cias</h1>");
    }
}
```

```

        out.println("Notícias cadastradas:<br/><br/>");
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"colocar a senha");
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM
tbnoticias");
            while (rs.next()) {
                out.println("<b>ID:</b> " + rs.getString("id")
+ "<br>");
                out.println("<b>Título:</b> " + rs.getString("titulo")
+ "<br>");
                out.println("<b>Data:</b> " + rs.getString("data")
+ "<br>");
                out.println("<b>Texto:</b> " + rs.getString("texto")
+ "<br>");
                out.println("<b>Autor:</b> " + rs.getString("autor")
+ "<br>");
                out.println("<br>");
            }
            conn.close();
        } catch (Exception e) {
            out.close();
        } finally {
            out.close();
        }

        out.println("</body>");
        out.println("</html>");
    }

    protected void doGet(HttpServletRequest request, HttpSer-
vletResponse response) throws ServletException, IOException
    {
        processRequest(request, response);
    }

    protected void doPost(HttpServletRequest request, Http-
ServletResponse response) throws ServletException, IOExcep-
tion {
        processRequest(request, response);
    }

    public String getServletInfo() {
        return "Short description";
    }
}

```

2º Servlet - GerenciaNoticiasSrv

Crie um novo arquivo **Servlet** com o nome **GerenciaNoticiasSrv** no pacote noticia. (caso haja dúvidas sobre a criação de arquivos Servlet, reveja como foi criado o primeiro arquivo. Deve ser executado a mesma seqüência de passos). Com o novo arquivo criado, apague todo o conteúdo deste **Servlet** e digite o seguinte código:

```
package noticias;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class GerenciaNoticiasSrv extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=iso-8859-1");
        PrintWriter out = response.getWriter();
        try {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Notícias Servlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Gerenciar Notícias</h1>");
            out.println("Notícias cadastradas:<br/><br/>");
            try {
                Class.forName("com.mysql.jdbc.Driver");
                Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"colocar a senha");
                Statement stmt = conn.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT id,
titulo FROM tbnoticias");
                out.println("<table width=\"100%\"
border=\"1\">");
                while (rs.next()) {
                    out.println("<tr>");
                    out.println("<td>" + rs.getString("id")
+ "</td>");
```

```

        out.println("<td> " + rs.getString("titulo")
+ "</td>");
        out.println("<td><a href=\"AlteraNoticiaSrv?id="
+ rs.getString("id") + "\">Alterar</a></td>");
        out.println("<td><a href=\"ExcluiNoticiaSrv?id="
+ rs.getString("id") + "\">Excluir</a></td>");
        out.println("</tr>");
    }
    out.println("</table>");
    out.println("<br/><a href=\"InsererNoticiaSrv
\">Clique aqui</a> para inserir uma nova notícia.");
    conn.close();
} catch (Exception e) {
    out.close();
}
out.println("</body>");
out.println("</html>");
} finally {
    out.close();
}
}

protected void doGet (HttpServletRequest request, HttpServlet
letResponse response) throws ServletException, IOException {
    processRequest(request, response);
}

protected void doPost (HttpServletRequest request, HttpServlet
letResponse response) throws ServletException, IOException {
    processRequest(request, response);
}

public String getServletInfo() {
    return "Short description";
}
}

```

3º Servlet - InserirNoticiaSrv

Crie um novo arquivo **Servlet** com o nome **InserirNoticiaSrv** no pacote noticia. (caso haja dúvidas sobre a criação de arquivos Servlet, reveja como foi criado o primeiro arquivo. Deve ser executado a mesma sequência de passos). Com o novo arquivo criado, apague todo o conteúdo deste **Servlet** e digite o seguinte código:

```
package noticias;
```

```

import java.io.IOException;
import java.sql.*;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class InserirNoticiaSrv extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=iso-8859-1");
        PrintWriter out = response.getWriter();
        try {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Notícias Servlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Insere Notícia </h1>");
            out.println("<br/>");
            out.println("<form action=\"InserirNoticiaSrv\" method=\"POST\">");
            out.println("<p>Título: <input type=\"text\" name=\"txttitulo\" /></p>");
            out.println("<p>Data: <input type=\"text\" name=\"txtdata\" /></p>");
            out.println("<p>Texto: <textarea name=\"txttexto\"></textarea></p>");
            out.println("<p>Autor: <input type=\"text\" name=\"txtautor\" /></p>");
            out.println("<p><input type=\"submit\" value=\"Insere\" name=\"btninsere\" /></p>");
            out.println("</form>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=iso-8859-1");
        PrintWriter out = response.getWriter();
        try {

```

```

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
            out.println("<title>Not&iacute;cias Servlet</
title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Insere Not&iacute;cia</h1>");
        out.println("<br/>");
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"colocar a senha");
            Statement stmt = conn.createStatement();
            int rs = stmt.executeUpdate("INSERT
INTO tbnoticias(titulo, data, texto, autor) VAL-
UES(' + request.getParameter("txttitulo") + ',
' + request.getParameter("txtdata") + ', ' + re-
quest.getParameter("txttexto") + ', ' + request.
getParameter("txtautor") + ')");
            if (rs == 0) {
                out.println("Ocorreu um erro na
inserção.<br/>");
            } else {
                out.println("Notícia inserida com
sucesso.<br/>");
            }
            conn.close();
        } catch (Exception e) {
            out.close();
        }
        out.println("</body>");
        out.println("</html>");
    } finally {
        out.close();
    }
}

public String getServletInfo() {
    return "Short description";
}
}

```

4º Servlet - AlteraNoticiaSrv

Crie um novo arquivo **Servlet** com o nome **AlterarNoticiaSrv** no pacote *noticia*. (caso haja dúvidas sobre a criação de arquivos Servlet, reveja como foi

criado o primeiro arquivo. Deve ser executado a mesma sequência de passos).
Com o novo arquivo criado, apague todo o conteúdo deste **Servlet** e digite o seguinte código:

```
package noticias;

import java.io.IOException;
import java.sql.*;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class AlteraNoticiaSrv extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=iso-8859-1");
        PrintWriter out = response.getWriter();
        try {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Notícias Servlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Altera Notícia</h1>");
            out.println("<br/>");
            try {
                Class.forName("com.mysql.jdbc.Driver");
                Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"colocar a senha");
                Statement stmt = conn.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT *
FROM tbnoticias WHERE id=" + request.getParameter("id"));
                rs.next();
                out.println("<form action=\"AlteraNoticiaSrv\"
method =\"POST\">");
                out.println("<input type=\"hidden\" name=\"id\"
value=\"\" + request.getParameter("id") + \"\" >");
                out.println("<p>Titulo: <input type=\"text\"
name=\"txttitulo\" value=\"\" + rs.getString("titulo") + \"\"
/> </p> ");
                out.println("<p> Data: <input type=\"text\"
name=\"txtdata\" value=\"\" + rs.getString("data") + \"\"
/> </p> ");
```

```

        out.println(" <p> Texto: <textarea
name=\"txttexto\">" + rs.getString("texto") + "</textarea>
</p>");
        out.println(" <p> Autor: <input type=\"text\"
name=\"txtautor\" value=\"" + rs.getString("autor") + "\" />
</p> ");
        out.println("<p><input type=\"submit\"
value=\"Alterar\" name=\"btnalterar\" /> </p> ");
        out.println("</form>");
        conn.close();
    } catch (Exception e) {
        out.close();
    }
    out.println("</body>");
    out.println("</html>");
} finally {
    out.close();
}
}

```

```

protected void doPost(HttpServletRequest request, Http-
ServletResponse response) throws ServletException, IOExcep-
tion {
    response.setContentType("text/html;charset=iso-8859-1");
    PrintWriter out = response.getWriter();
    try {
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Notícias Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Alterar Notícia</h1>");
        out.println("<br/>");
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn
                = DriverManager.getConnection("jdbc:mysql://
localhost/bdnoticias", "root", "colocar a senha");
            Statement stmt = conn.createStatement();
            int rs = stmt.executeUpdate("UPDATE tb-
noticias SET titulo='" + request.getParameter("txttitulo")
+ "', data='" + request.getParameter("txtdata") + "', tex-
to='" + request.getParameter("txttexto") + "', autor='" +
request.getParameter("txtautor") + "' WHERE id=" + request.
getParameter("id") + ";"");
            if (rs == 0) {
                out.println("Ocorreu um erro na
alteração.<br/>");
            }
        }
    }
}

```



```

        } else {
            out.println("Notícia alterada com
sucesso.<br/>");
        }
        conn.close();
    } catch (Exception e) {
        out.close();
    }
    out.println("</body>");
    out.println("</html>");
} finally {
    out.close();
}
}

public String getServletInfo() {
    return "Short description";
}
}

```

5º Servlet - ExcluiNoticiaSrv

Crie um novo arquivo **Servlet** com o nome **ExcluiNoticiaSrv** no pacote noticia. (caso haja dúvidas sobre a criação de arquivos Servlet, reveja como foi criado o primeiro arquivo. Deve ser executado a mesma sequência de passos). Com o novo arquivo criado, apague todo o conteúdo deste **Servlet** e digite o seguinte código:

```

package noticias;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ExcluiNoticiaSrv extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOEx-
ception {
        response.setContentType("text/html;charset=iso-8859-1");
        PrintWriter out = response.getWriter();
        try {
            out.println("<!DOCTYPE html>");

```

```

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Notícias Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Exclui Notícia</h1>");
        out.println("<br/>");
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"colocar a senha");
            Statement stmt = conn.createStatement();
            int rs = stmt.executeUpdate("DELETE FROM tb-
noticias WHERE id=" + request.getParameter("id") + ";");
            if (rs == 0) {
                out.println("Ocorreu um erro na
exclusão.<br/>");
            } else {
                out.println("Notícia excluída com
sucesso.<br/>");
            }
            conn.close();
        } catch (Exception e) {
            out.close();
        }
        out.println("</body>");
        out.println("</html>");
    } finally {
        out.close();
    }
}

protected void doGet(HttpServletRequest request, HttpServ-
letResponse response) throws ServletException, IOException {
    processRequest(request, response);
}

protected void doPost(HttpServletRequest request, HttpServ-
letResponse response) throws ServletException, IOException {
    processRequest(request, response);
}

public String getServletInfo() {
    return "Short description";
}
}

```

6º arquivo - index.html

Clique com o **botão direito** sob a pasta “*Páginas Web*” que está dentro da estrutura hierárquica de pastas da aplicação Web denominada **NoticiasServlet**. Na sequência, selecione a opção **Novo** → **HTML**. Na tela que abrir, atribua no campo “*Nome do Arquivo HTML*”, o nome **index** e clique em **Finalizar**. Com o novo arquivo criado, apague todo o conteúdo deste arquivo HTML e digite o seguinte código:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="iso-8859-1">
    <title> Notícias Servlet </title>
  </head>
  <body>
    <h1> Notícias Servlet </h1>
    Escolha a opção abaixo: <br/>
    <ul>
      <li> <a href="VisualizaNoticiasSrv"> Visualizar
notícias </a> </li>
      <li> <a href="GerenciaNoticiasSrv"> Gerenciar
notícias </a> </li>
    </ul>
  </body>
</html>
```

Pronto, a aplicação Web está quase pronta! Agora precisa configurar o **arquivo web.xml** dentro da pasta **WEB-INF**.

Ao localizar o arquivo **web.xml**, clique duas vezes sob o arquivo para abrir uma tela de configuração do arquivo como apresentado na Figura 110 a seguir.

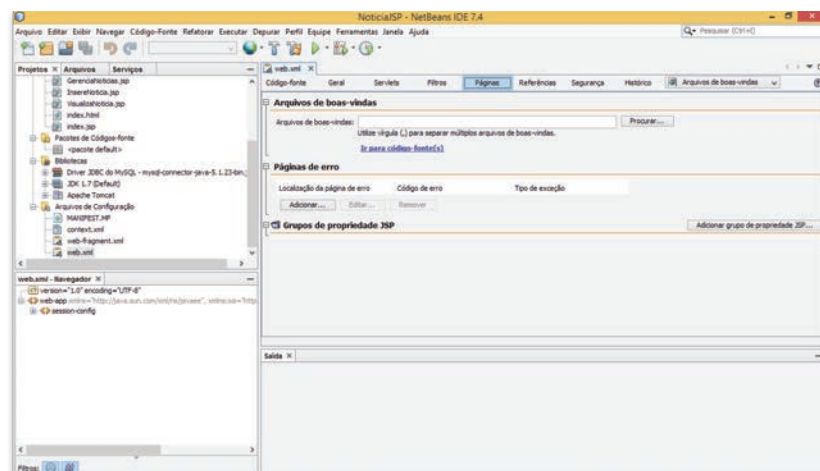


Figura 110 - Configurando o arquivo web.xml

No campo denominado “Arquivo de boas vindas” coloque o nome da página **index.html**. Para isso, clique na opção “Procurar...”, selecione o arquivo **index.html** e clique em **Selecionar o Arquivo**.

Para finalizar o exemplo precisa ser criado o Banco de Dados, então procure o aplicativo **MySQL Command Line Client**.

O aplicativo **MySQL Command Line Client** permite acessar o Banco de Dados e inserir comando SQL para criação de tabelas, geração de consultas e etc. Trata-se de um programa executado no **Command Prompt** do Windows como apresentado na Figura 111.

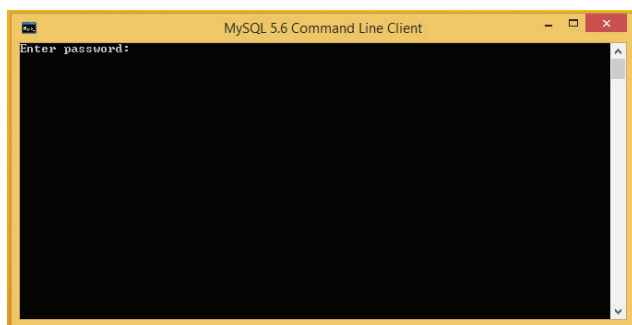


Figura 111 – MySQL Command Line Client

Ao executar o aplicativo do MySQL é solicitado a inserção da senha definida no momento da instalação do Banco de Dados MySQL.

Após digitar a senha, clique na tecla **enter** e a área para inserir comando SQL é liberada como apresentado na Figura 112.

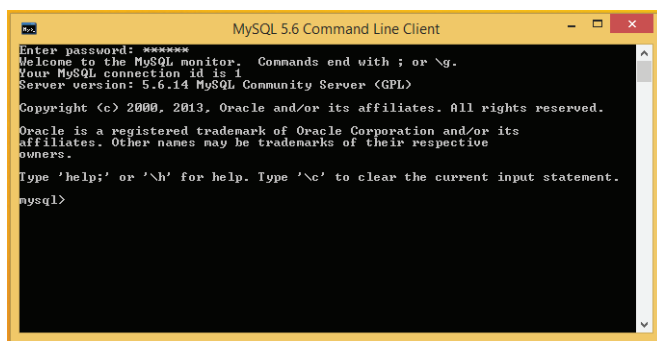
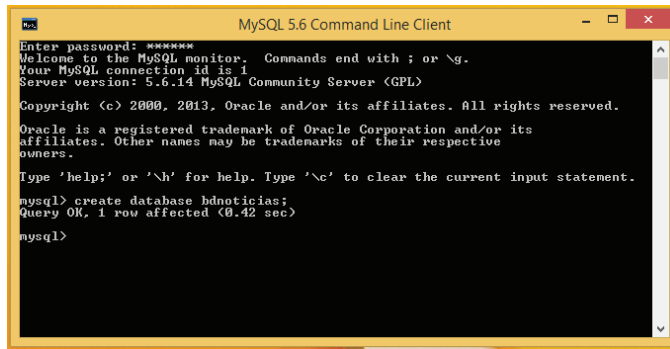


Figura 112 - MySQL Command Line Client

No terminal igual ao apresentado na Figura 112, digite a linha de comando “`create database bdnoticias;`”. Tal linha define a criação de uma base de dados com o nome **bdnoticias**.

Após digitar o comando clique na tecla **enter** e a base de dados é gerada, como apresentado na Figura 113.



```
MySQL 5.6 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

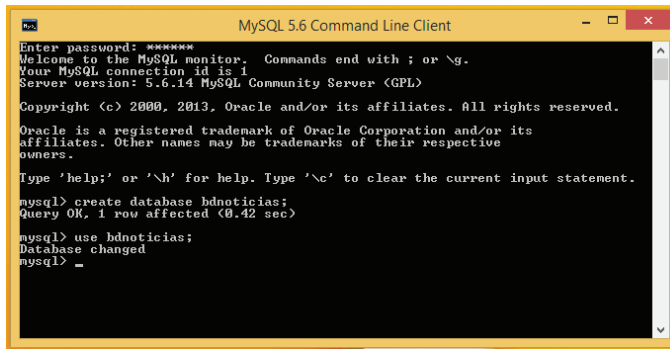
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database bdnoticias;
Query OK, 1 row affected (0.42 sec)

mysql>
```

Figura 113 - Criação da base de dados

Com o banco criado é necessário informar ao SGBD qual a base de dados que vai ser utilizada. Assim, use o código “use bdnoticias;” para indicar qual a base vai ser usada a partir desse momento no **MySQL Command Line Client** como apresentado na Figura 114.



```
MySQL 5.6 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database bdnoticias;
Query OK, 1 row affected (0.42 sec)

mysql> use bdnoticias;
Database changed
mysql> _
```

Figura 114 - Acessando o Banco de Dados criado.

Definido qual o Banco de Dados que vai ser criado as tabelas do sistema de notícias, insira a seguinte linha de comando no terminal:

```
CREATE TABLE `bdnoticias`.`tbnoticias` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `data` datetime NOT NULL,
  `titulo` varchar(100) NOT NULL,
  `texto` text NOT NULL,
  `autor` varchar(100) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

A linha de comando está escrita em **linguagem SQL** e cria uma **tabela de notícias**. A tabela de notícias está com o nome **tbnoticias** e contém um campo identificador auto-incremento, a data de postagem, o título da notícia, o texto e o autor. Após a criação da tabela, digite o comando “quit;” para sair do terminal do **MySQL**.

Voltando ao NetBeans, clique com o *botão direito* sob o projeto “**Noticias-Servlet**” e selecione a opção **Executar**.

O projeto é executado, abrindo inicialmente a página **index.html**. Com a página **index.html** aberta no navegador, clique nos **links** para **navegar**, **inserir**, **alterar** e **consultar** os dados.

5.3.3 Aplicação com BD + JSP + CSS + JavaScript

A aplicação a ser desenvolvida neste exemplo é o mesmo sistema de gerenciamento comentado nos tópicos 5.3.2 e 5.3.4. A diferença neste exemplo é que vai ser usado os recursos de folha de estilo e JavaScript comentados nas primeiras unidades.

Então, para iniciar crie um projeto clicando no menu **Arquivo** → **Novo Projeto** como apresentado na Figura 115.

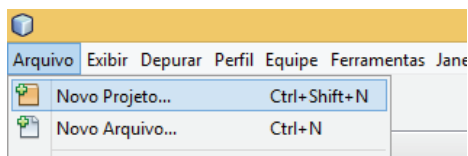


Figura 115 - Novo projeto

Ao clicar na opção “**Novo Projeto**” surge uma tela que permite selecionar qual o tipo de projeto e linguagem de programação.

A aplicação a ser desenvolvida é para a Web. Sendo assim, selecione a opção **Java Web** e a opção **Aplicação Web**. Na sequência, clique no botão **Próximo** e informe o nome do projeto que neste caso deve ser **NoticiasCompleta**.

Na próxima tela configure o servidor Tomcat, pule a etapa de seleção de *framework* e por fim clique em finalizar.

Todos os passos mencionados anteriormente segue o que foi apresentado nos tópicos 5.3.2 e 5.3.3. Lembre-se de configurar neste novo projeto a biblioteca do MySQL que permite a conexão com o Banco de Dados.

Para a aplicação funcionar precisa ser criado alguns arquivos JSPs, crie novos JSPs seguindo o código dos arquivos a seguir:

1º Arquivo JSP – index.jsp

O nome do arquivo é **index.jsp** e o conteúdo contempla o seguinte código:

```

<%@page contentType="text/html" pageEncoding="iso-8859-1"%>
<%@page import="java.sql.*" %>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="iso-8859-1" />
    <title>Sistema de Gerenciamento de Notícias</title>
    <!--Garante o suporte correto do HTML5 para o Internet
Explorer-->
    <!--[if lt IE 9]>
      <script src="http://html5shiv.googlecode.com/svn/
trunk/html5.js"></script>
    <![endif]-->
    <link      rel="stylesheet"      href="_css/estilo.css"
type="text/css"/>
  </head>
<body>
  <header>
    <span id="logotipo"></span>
    <nav>
      <ul id="menu">
        <li><a href="index.jsp">Notícias</a></li>
        <li><a href="GerenciaNoticias.jsp">Gerenciar</
a></li>
        <li><a href="faleconosco.html">Fale
Conosco</a></li>
      </ul>
    </nav>
  </header>
  <section>
  <h1>Notícias</h1>
  <%
    try {
      Class.forName("com.mysql.jdbc.Driver");
      Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"senha do banco");
      Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT * FROM
tbnoticias");
      while (rs.next()) {
%>
        <article>
          <table>
            <thead>
              <tr>
                <th>Título:</th>
                <th><%= rs.getString("titulo")%></th>

```

```

        <th> / Data:</th>
        <th><%= rs.getString("data") %></th>
    </tr>
</thead>
<tr>
    <td>Autor:</td>
    <td colspan="3"><%= rs.getString("autor") %></
td>
    </tr>
    <tr>
        <td>Texto:</td>
        <td colspan="3"><%= rs.getString("texto") %></
td>
    </tr>
</table>
<br>
</article>

<%
    }
    conn.close();
} catch (Exception e) {
    //ocorreu um erro
}
%>

</section>
</body>
</html>

```

2º Arquivo JSP – GerenciaNoticias.jsp

O nome do arquivo é **GerenciaNoticias.jsp** e o conteúdo contempla o seguinte código:

```

<%@page contentType="text/html" pageEncoding="iso-8859-1"%>
<%@page import="java.sql.*" %>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="iso-8859-1" />
        <title>Sistema de Gerenciamento de Notícias</title>
        <!--Garante o suporte correto do HTML5 para o Internet
Explorer-->
        <!--[if lt IE 9]>
            <script src="http://html5shiv.googlecode.com/svn/
trunk/html5.js"></script>

```



```

        <![endif]-->
        <link rel="stylesheet" href="_css/estilo.css" type="text/
css"/>
    </head>
<body>
    <header>
        <span id="logotipo"></span>
        <nav>
            <ul id="menu">
                <li><a href="index.jsp">Notícias</a></li>
                <li><a href="GerenciaNoticias.jsp">Gerenciar</
a></li>
                <li><a href="faleconosco.html">Fale
Conosco</a></li>
            </ul>
        </nav>
    </header>
    <section>
        <h1>Gerenciar</h1>
        <article>
            <%
            try {
                Class.forName("com.mysql.jdbc.Driver");
                Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"senha do banco");
                Statement stmt = conn.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT id,
titulo FROM tbnoticias");
            >
            <table class="gerencia">
            <%
                while (rs.next()) {
            >
                <tr>
                    <td> <%= rs.getString("id")%> </td>
                    <td> <%= rs.getString("titulo")%> </td>
                    <td> <a href="AlteraNoticia.jsp?id=<%=
rs.getString("id")%>"> Alterar </a> </td>
                    <td> <a href="ExcluiNoticia.jsp?id=<%=
rs.getString("id")%> "> Excluir </a> </td>
                </tr>
            <%
                }
            >
            </table>
            <br/>

```

` Clique aqui para inserir uma nova notícia.`

```
<%
    conn.close();
} catch (Exception e) {
    //ocorreu um erro
}
%>
</article>
</section>

</body>
</html>
```

3º arquivo JSP – InsererNoticia.jsp

O nome do arquivo é **InsererNoticia.jsp** e o conteúdo contempla o seguinte código:

```
<%@page contentType="text/html" pageEncoding="iso-8859-1"%>
<%@page import="java.sql.*" %>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="iso-8859-1" />
    <title>Sistema de Gerenciamento de Notícias</title>
    <!--Garante o suporte correto do HTML5 para o Internet Explorer-->
    <!--[if lt IE 9]>
      <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
    <![endif]-->
    <link rel="stylesheet" href="_css/estilo.css" type="text/css"/>
  </head>
  <body>
    <header>
      <span id="logotipo"></span>
      <nav>
        <ul id="menu">
          <li><a href="index.jsp">Notícias</a></li>
          <li><a href="GerenciaNoticias.jsp">Gerenciar</a></li>
          <li><a href="faleconosco.html">Fale Conosco</a></li>
        </ul>
```

```

        </nav>
</header>
<section>
    <article>

<%
    if (request.getMethod() != "POST") {
%>

        <h1>Insere Notícia</h1>
        <br/>
        <form action="InsereNoticia.jsp" method="POST">
            <p>
                <label for="txttitulo">Título:</label>
                <input id="txttitulo" type="text" name="txttitulo"
class="texto"/>
            </p>
            <p>
                <label for="txtdata">Data:</label>
                <input id="txtdata" type="text" name="txtdata"
class="texto"/>
            </p>
            <p>
                <label for="txttexto">Texto:</label>
                <textarea id="txttexto" name="txttexto"
rows="10" cols="40" class="textarea"></textarea>
            </p>
            <p>
                <label for="txtautor">Autor:</label>
                <input id="txtautor" type="text" name="txtautor"
class="texto"/>
            </p>
            <p>
                <label></label>
                <input type="submit" value="Insere"
name="btninsere" class="botao"/>
            </p>
        </form>

<%
    }
    else
    {

        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"senha do banco");

```

```

        Statement stmt = conn.createStatement();
        int rs = stmt.executeUpdate("INSERT INTO
tbnoticias(titulo, data, texto, autor) VALUES('" + re-
quest.getParameter("txttitulo") + " , '" + re-
quest.getParameter("txtdata") + " , '" + request.
getParameter("txttexto") + " , '" + request.
getParameter("txtautor") + "')");
        if (rs == 0) {
%>
                Ocorreu um erro na inserção.<br/>
<%
        }
        else
        {
%>
                Notícia inserida com sucesso.<br/>
<%
        }
        conn.close();
    } catch (Exception e){
//ocorreu um erro
    }
}
%>

</article>
</section>
</body>
</html>

```

4º arquivo JSP – AlteraNoticia.jsp

O nome do arquivo é **AlterarNoticia.jsp** e o conteúdo contempla o seguinte código:

```

<%@page contentType="text/html" pageEncoding="iso-8859-1"%>
<%@page import="java.sql.*" %>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="iso-8859-1" />
        <title>Sistema de Gerenciamento de Notícias</title>
        <!--Garante o suporte correto do HTML5 para o Internet
Explorer-->
        <!--[if lt IE 9]>
            <script src="http://html5shiv.googlecode.com/svn/
trunk/html5.js"></script>
        <![endif]-->

```

```

    <link rel="stylesheet" href="_css/estilo.css" type="text/
css"/>
    </head>
<body>
    <header>
        <span id="logotipo"></
span>
            <nav>
                <ul id="menu">
                    <li><a href="index.jsp">Notícias</a></li>
                    <li><a href="GerenciaNoticias.jsp">Gerenciar</
a></li>
                    <li><a href="faleconosco.html">Fale
Conosco</a></li>
                </ul>
            </nav>
        </header>
        <section>
            <article>

                <% if (request.getMethod() != "POST") {
                    try {
                        Class.forName("com.mysql.jdbc.Driver");
                        Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"senha do banco");
                        Statement stmt = conn.createStatement();
                        ResultSet rs = stmt.executeQuery("SELECT *
FROM tbnoticias where id =" + request.getParameter("id")+";");
                        while (rs.next()) {
                            %>
                                <form action="AlterarNoticia.jsp"
method="POST">
                                    <p>
                                        <label for="txttitulo">Título:</
label>
                                            <input type="text" name="txttitulo"
value="<%= rs.getString("titulo")%>" class="texto"/>
                                                </p>
                                                <p>
                                                    <label for="txtdata">Data:</
label>
                                                        <input type="text" name="txtdata"
value="<%= rs.getString("data")%>" class="texto"/>
                                                            </p>
                                                            <p>
                                                                <label for="txttexto">Texto:</
label>

```

```

        <textarea name="txttexto" rows="10"
cols="40" class="textarea"><%= rs.getString("texto")%></
textarea>

        </p>
        <p>
            <label for="txtautor">Autor:</
label>

            <input type="text" name="txtautor"
value="<%= rs.getString("autor")%>" class="texto"/>
            </p>
            <p>
                <label></label>
                <input type="submit"
value="Alterar" name="btalterar" class="botao" />
            </p>
            <input type="hidden" name="id"
value="<%= rs.getString("id")%>" />
        </form>
    <%
        }
        conn.close();
    } catch (Exception e) {
        //ocorreu um erro
    }

    } else {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"senha do banco");

            Statement stmt = conn.createStatement();
            int rs = stmt.executeUpdate("UPDATE
tbnoticias SET titulo=' " + request.getParameter("txttitulo")
+ "' , data=' " + request.getParameter("txtdata") + "' , tex-
to=' " + request.getParameter("txttexto") + "' , autor=' " +
request.getParameter("txtautor") + "' WHERE id=" + request.
getParameter("id") + ";"");
            if (rs == 0) {
                %>
                    Ocorreu um erro na alteraao.<br/>
                <%
                    } else {
                %>
                    Notícia alterada com sucesso.<br/>
                <%
                    }
                    conn.close();
                } catch (Exception e) {
                    // trata erro

```

```

        }
    }
%>

</article>
</section>
</body>
</html>

```

5º arquivo JSP – ExcluiNoticia.jsp

O nome do arquivo é **ExcluiNoticia.jsp** e o conteúdo contempla o seguinte código:

```

<%@page contentType="text/html" pageEncoding="iso-8859-1"%>
<%@page import="java.sql.*" %>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="iso-8859-1" />
    <title>Sistema de Gerenciamento de Notícias</title>
    <!--Garante o suporte correto do HTML5 para o Internet
Explorer-->
    <!--[if lt IE 9]>
      <script src="http://html5shiv.googlecode.com/svn/
trunk/html5.js"></script>
    <![endif]-->
    <link rel="stylesheet" href="_css/estilo.css" type="text/
css"/>
  </head>
  <body>
    <header>
      <span id="logotipo"></
span>
      <nav>
        <ul id="menu">
          <li><a href="index.jsp">Notícias</a></li>
          <li><a href="GerenciaNoticias.jsp">Gerenciar</
a></li>
          <li><a href="faleconosco.html">Fale
Conosco</a></li>
        </ul>
      </nav>
    </header>
    <section>
      <article>
        <h1>Excluir</h1>
        <%

```

```

        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.
getConnection("jdbc:mysql://localhost/bdnoticias", "root",
"senha do banco");
            Statement stmt = conn.createStatement();
            int rs = stmt.executeUpdate("DELETE FROM tb-
noticias WHERE id=" + request.getParameter("id") + ";");
            if (rs == 0) {
                %>
                Ocorreu um erro na exclusão.<br/>
                <%> else { %>
                Notícia excluída com sucesso.<br/>
                <% } conn.close();
            } catch (Exception e) {
                //tratar erro
            }
            %>
        } </article>
    </section>
</body>
</html>

```

6º arquivo JSP – faleconosco.html

O nome do arquivo é **faleconosco.html** e o conteúdo contempla o seguinte código:

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title>Sistema de Gerenciamento de Notícias</title>
        <!--Garante o suporte correto do HTML5 para o Inter-
net Explorer-->
        <!--[if lt IE 9]>
            <script src="http://html5shiv.googlecode.com/svn/
trunk/html5.js"></script>
        <![endif]-->

        <link rel="stylesheet" href="_css/estilo.css"
type="text/css"/>
        <!-- Código JavaScript para obrigar que o usuário só
informe algum valor nos campos -->
        <script>
            /*
            *

```



```

        * A função validar tem como corpo do código duas va-
riáveis locais com o nome
        * de nome e mensagem.
        * Cada variável recebe o valor do formulário, para
isso é usado o comando
        * document.getElementById(nome do ID do campo) se-
guido do ponto value.
        * Para realizar a comparação é usado a estrutura
condicional IF e o
        * operador de comparação igual == .
        *
        */

        function validar() {
            var nome = document.getElementById("nome").
value;
            var mensagem = document.getElementById("mensagem").
value;

            if (nome == "") {
                alert("Deve ser informado um valor para
o nome!");
            } else if (mensagem == "") {
                alert("Deve ser informado um valor para
a mensagem!");
            }
        }
    </script>

</head>
<body>
    <header>
        <span id="logotipo"></
span>
        <nav>
            <ul id="menu">
                <li><a href="index.jsp">Notícias</a></li>
                <li><a href="GerenciaNoticias.jsp">Gerenciar</
a></li>
                <li><a href="faleconosco.html">Fale Conosco</
a></li>
            </ul>
        </nav>
    </header>
    <section>
        <article>
            <form action="#" method="POST">
                <fieldset>
                    <legend>Entre em contato com a equipe:
</legend>

```

```

        <p>
            <label for="nome">Nome:</label>
            <input type="text" id="nome"
name="nome" class="texto" />
        </p>
        <p>
            <label for="assunto">Assunto:</
label>
            <select name="assunto">
                <option selected>Notícia</
option>
                <option>Solicitar retirada
de notícia</option>
            </select>
        </p>
        <p>
            <label for="mensagem">Mensagem:</
label>
            <textarea class="textarea" name="mensagem"
rows="10" cols="40" id="mensagem"></textarea>
        </p>
        <p>
            <label></label>
            <input type="submit" value="enviar"
onclick="validar();" class="botao"/>
        </p>
    </fieldset>
</form>
</article>
</section>
</body>
</html>

```

Dentro da aplicação web **“NoticiasCompleta”** crie uma pasta denominada **_css** e dentro da pasta crie um arquivo com o nome *estilo.css*.

```

/*
Define a margem e o espaçamento com o valor igual a zero.

```

A propriedade `margin` refere-se a margem e ao aplicar o valor igual a zero tanto no topo, embaixo, lado direito e esquerdo da imagem a margem é igual a zero.

A propriedade `padding` define um espaçamento do conteúdo com as bordas ao retorno do elemento. Neste caso, quando definido 0 significa que todos os elementos tem o padding tanto em cima, abaixo, lado direito e lado esquerdo igual a zero.

```

*/

```

```

* {
    margin:0;
    padding:0;
}

/*
Define a fonte da página em 88% e fonte igual a Arial, Helve-
tica e Sans-serif
*/
body {
    font:88% Arial, Helvetica, sans-serif;
}

/*
A propriedade text-decoration pertencente a tag "a" que define
um link no documento HTML, quando atribuída com o valor none
retira o sublinhado.
*/
a {
    text-decoration: none;
}

/*
A propriedade list-style-type retira os bullets de uma lista.
Entende-se por bullets o desenho ou símbolo que fica ao lado
esquerdo de uma lista.
*/
ul,ol {
    list-style-type:none;
}

/*
A propriedade width define a largura do cabeçalho representado
pela tag "header" do HTML5 em 757px de largura.

A propriedade margin define as margens desse elemento. Enten-
de-se margem como sendo o espaçamento entre o topo, lado di-
reito, lado esquerdo e abaixo de um elemento. Quando a margem
está automática significa que o conteúdo fica centralizado na
página.
*/
header {
    width: 757px;
    margin: auto;
}

/*

```

A propriedade background-color define a cor de fundo do elemento nav em cinza. Assim, as cores em folha de estilo (CSS) são definidas em Hexadecimais.

A propriedade height define a altura em 25px, dessa forma define uma espessura a barra criada em folha de estilo.

A propriedade margin-top define o valor de 20px, ou seja está à 20px em relação ao topo do navegador neste caso.

```
*/
nav {
    background-color: #6d7276;
    height: 25px;
    margin-top: 20px;
}
```

```
/*
A propriedade float define o alinhamento do elemento que está com o ID #logotipo. Assim, quando o valor da propriedade é igual a left indica que o elemento é lançado à esquerda.
```

```
*/
#logotipo {
    float: left;
}
```

```
/*
Uma lista é formada por lista de itens e cada item deve ser alinhado à esquerda. Quando a propriedade left é atribuída ao elemento li significa esse comportamento.
```

```
*/
#menu li {
    float:left;
}
```

```
/*
Quando encontrar um ID definido como #menu e que tenha a tag a é aplicado as seguintes regras:
```

- Atribuir um padding de 5px margem top;
- Atribuir um padding de 5px margem bottom;
- Atribuir um padding de 12px margem right;
- Atribuir um padding de 12px margem left;
- Atribuir a cor em branco;
- Atribuir a aparência em bloco;

```
*/
#menu a {
    padding: 5px 12px;
    color: #FFFFFF;
    display:block;
```

```

}

/*
Quando o usuário passa o mouse em cima do link aciona a
propriedade hover. Assim, a cor da fonte deve ser preta
(#000000), a cor de fundo branca (#FFFFFF) e deve ser apre-
sentado como bloco (display:block). */

#menu a:hover {
    background-color:#FFFFFF;
    color:#000000;
    display:block;
}

/*
Configurando a posição dos campos de entrada do Texto
*/

/*
A propriedade width define a largura da tag section em 757px.
A propriedade margin define as margens esquerda, direita, topo
e abaixo com valores automáticos.
A propriedade margin-top define a margem do topo igual a
50px.
*/
section {
    width: 757px;
    margin: auto;
    margin-top: 50px;
}

/*
A propriedade padding define um espaçamento do conteúdo com as
bordas ao retorno do elemento. Neste caso, quando definido 5
significa que o elemento tem o padding tanto em cima, abaixo,
lado direito e lado esquerdo igual a cinco.
*/
p {
    padding: 5px;
}

/*
A propriedade text-align define o alinhamento do texto para
direita, esquerda e centro. No contexto que está sendo tra-
tado o texto está sendo alinhado a direita.

A propriedade width estabelece a largura, neste caso está
sendo definido o tamanho de 75px do elemento label.

```

A propriedade `vertical-align` define que o conteúdo deve ser alinhado no topo do elemento ou alinhado na parte inferior do elemento. No exemplo, o texto está alinhado verticalmente ao topo.

A propriedade `display` simula o comportamento de um elemento em relação a forma como o elemento deve se comportar. Ou seja, o elemento `label` é um elemento `in-line` não provocando quebras de linha aos elementos que seguem o elemento `label`. No entanto para conseguir aplicar o tamanho `width` precisa ser definido a propriedade `inline-block`.

```
*/  
label {  
    text-align: right;  
    width: 75px;  
    vertical-align: top;  
    display: inline-block;  
}
```

/*
A propriedade `width` define a largura de 400px de uma tag `input` que tenha uma classe definida como sendo texto.

A propriedade `font-size` modifica o tamanho da fonte presente no elemento `input`. A unidade em está relacionado ao tamanho da letra M em maiúsculo sendo variável assim como a definição em porcentagem.

```
*/  
input.texto {  
    font-size:1em;  
    width: 400px;  
}
```

/*
A propriedade `padding` define um espaçamento do conteúdo com as bordas ao retorno do elemento. Neste caso, quando definido 5 significa que todos os elementos tem o padding tanto em cima, abaixo, lado direito e lado esquerdo igual a cinco.

A propriedade `color` define a cor do texto pertencente ao elemento `legend` em azul.

```
*/  
h1, legend {  
    padding: 5px;  
    color: #006AD6;  
}
```

```
input.botao {
```

```

        width: 100px;
        height: 30px;
    }

/*
A propriedade border-collapse transforma bordas duplas em bor-
das simples.
*/
table.gerencia {
    width: 100%;
    border-collapse: collapse;
    border: 1px solid #000000;
}

table.gerencia tr {
    border-bottom: 1px solid #000000;
}

table.gerencia tr td {
    padding: 5px;
}

```

Pronto, a aplicação Web está quase pronta! Agora precisa configurar o **arquivo web.xml** dentro da pasta **WEB-INF**.

Ao localizar o arquivo **web.xml**, clique duas vezes sob o arquivo para abrir uma tela de configuração do arquivo como apresentado na Figura 116 a seguir.

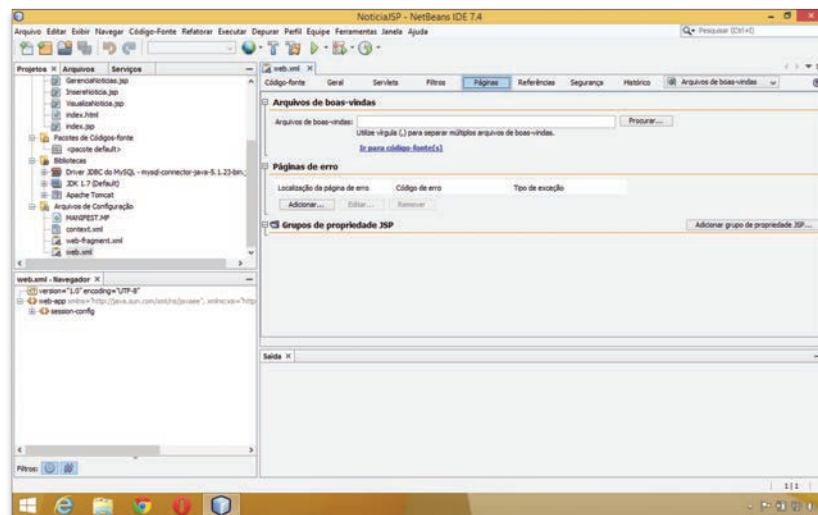


Figura 116 - Configurando o arquivo web.xml

No campo denominado “*Arquivo de boas vindas*” coloque o nome da página **index.jsp**. Para isso, clique na opção “**Procurar...**”, selecione o arquivo **index.jsp** e clique em **Selecionar o Arquivo**.

DICA

Caso o arquivo **web.xml** não exista na estrutura da **aplicação Web**, o desenvolvedor pode criar este arquivo.

Para criar o arquivo **web.xml**, basta clicar com o botão direito sob a **pasta WEB-INF** e **selecionar OUTROS**, como apresentado na Figura 117.

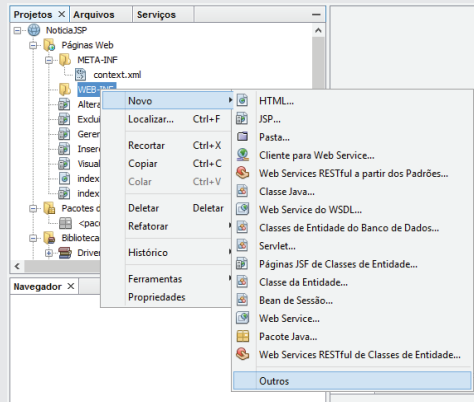


Figura 117 - Arquivo web.xml

Ao clicar em outros, uma tela é aberta solicitando ao desenvolvedor a definição para qual tipo de arquivo deve ser gerado. Assim, selecione a opção *“Descriptor de Implantação Padrão (web.xml)”* que está na **categoria Web**, como apresentado na Figura 118.

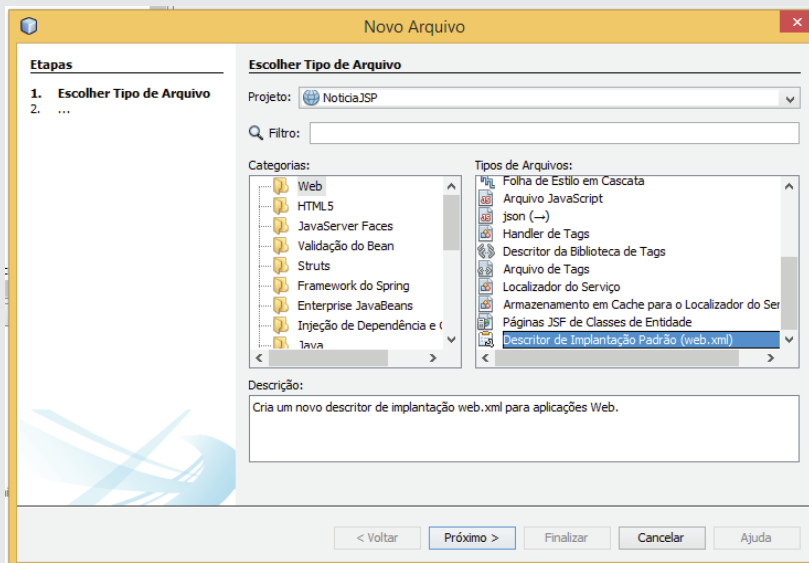


Figura 118 - Criar arquivo web.xml

Após selecionar o tipo de arquivo a ser criado, clique na opção **Próximo** e depois em **Finalizar**, como apresentado na Figura 119.

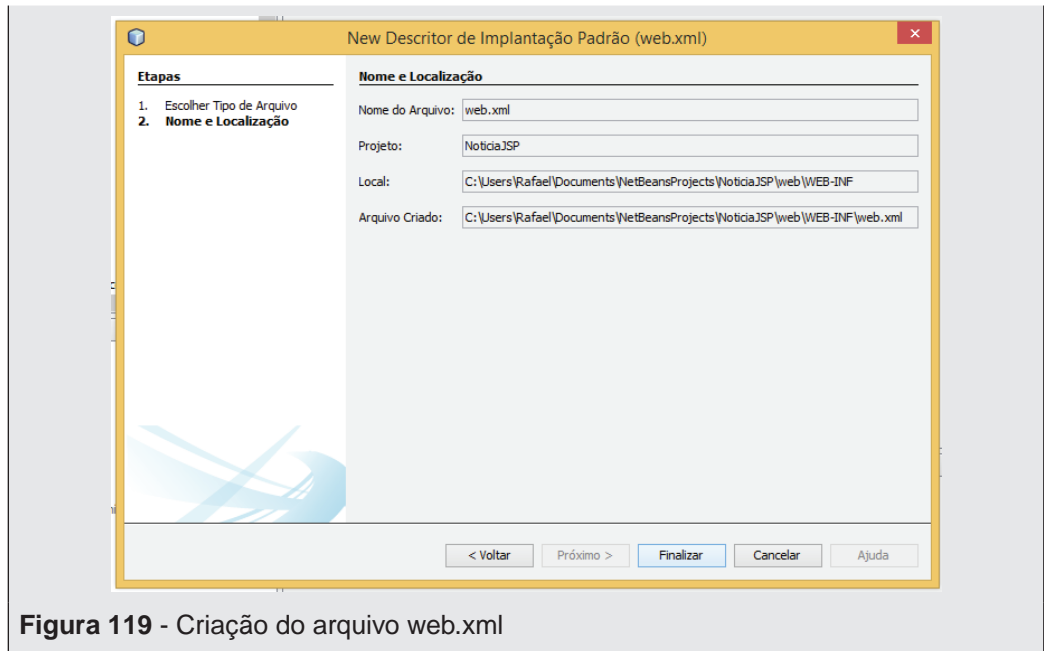


Figura 119 - Criação do arquivo web.xml

Para finalizar o exemplo precisa ser criado o Banco de Dados, então procure o aplicativo **MySQL Command Line Client**.

O aplicativo **MySQL Command Line Client** permite acessar o Banco de Dados e inserir comando SQL para criação de tabelas, geração de consultas e etc. Trata-se de um programa executado no **Command Prompt** do Windows como apresentado na Figura 120.

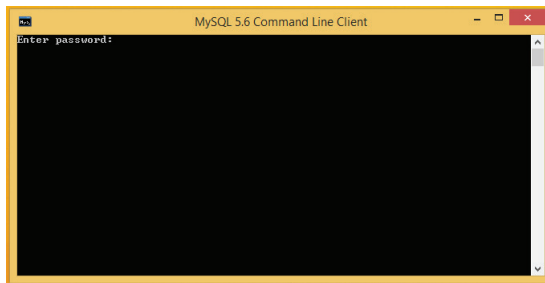


Figura 120 – MySQL Command Line Client

Ao executar o aplicativo do MySQL é solicitado a inserção da senha definida no momento da instalação do Banco de Dados MySQL.

Após digitar a senha, clique na tecla **enter** e a área para inserir comando SQL é liberada como apresentado na Figura 121.

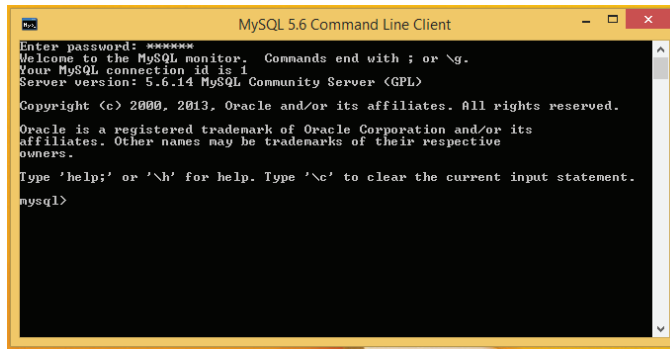


Figura 121 - MySQL Command Line Client

No terminal, igual ao apresentado na Figura 121, digite a a linha de comando “create database bdnoticias;”. Tal linha define a criação de uma base de dados com o nome **bdnoticias**.

Após digitar o comando clique na tecla **enter** e a base de dados é gerada, como apresentado na Figura 122.

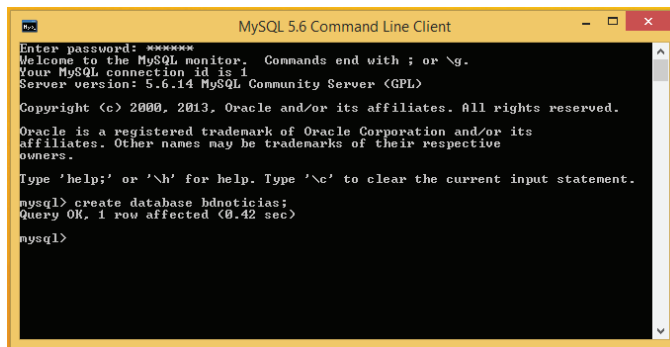


Figura 122 - Criação da base de dados

Com o banco criado é necessário informar qual a base de dados que vai ser utilizada. Assim, use o código “use bdnoticias;” para indicar qual a base vai ser usada a partir desse momento no **MySQL Command Line Client** como apresentado na Figura 123.

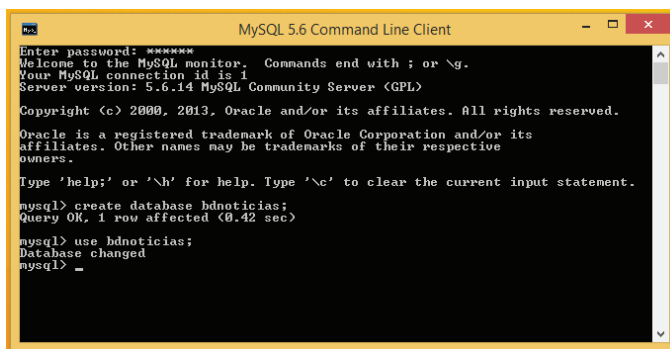


Figura 123 - Acessando o banco de dados criado.

Definido qual o Banco de Dados que vai ser criado as tabelas do sistema de notícias, insira a seguinte linha de comando no terminal:

```
CREATE TABLE `bdnoticias`.`tbnoticias` (  
  `id` int(10) unsigned NOT NULL auto_increment,  
  `data` datetime NOT NULL,  
  `titulo` varchar(100) NOT NULL,  
  `texto` text NOT NULL,  
  `autor` varchar(100) NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

A linha de comando está escrita em **linguagem SQL** e cria uma **tabela de notícias**. A tabela de notícias está com o nome **tbnoticias** e contém um campo identificador auto-incremento, a data de postagem, o título da notícia, o texto e o autor. Após a criação da tabela, digite o comando “quit;” para sair do terminal do **MySQL**.

Voltando ao NetBeans, clique com o *botão direito* sob o projeto “**Noticias-Completa**” e selecione a opção **Executar**.

O projeto é executado, abrindo inicialmente a página **index.jsp**. Com a página **index.jsp** aberta no navegador, clique nos **links** para navegar.

Ao executar a aplicação no **navegador Internet Explorer** por exemplo, o sistema de gerenciamento apresenta as notícias logo na página inicial (*se tiver alguma cadastrada*), como apresentado na Figura 124.



Figura 124 - Visualizar Notícias

Além de visualizar a notícia, o sistema de gerenciamento permite inserir novas notícias, alterar e excluir na opção gerenciar, como apresentado na Figura 125.

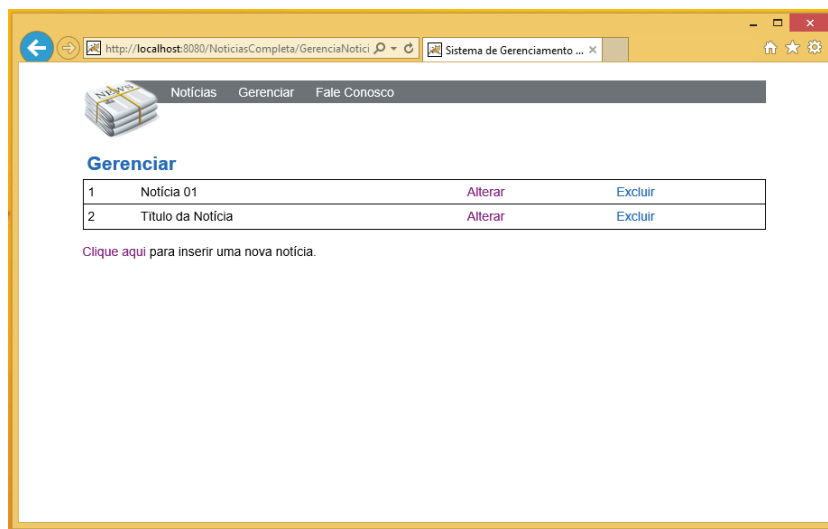


Figura 125 - Gerenciar notícias

Ao clicar na opção **“Clique aqui”** apresentada na Figura 125 é possível inserir uma nova notícia, como apresentado na Figura 126.

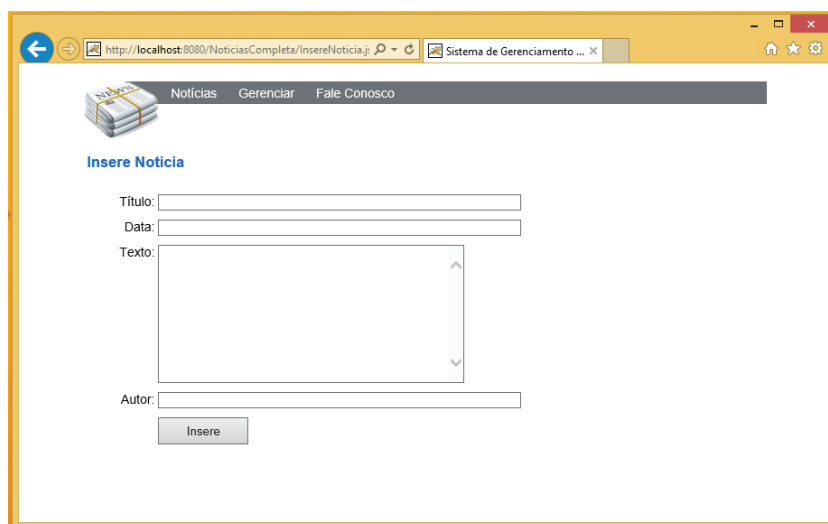


Figura 126 - Inserir uma nova notícia

Na Figura 125 que apresenta a tela de gerenciamento é possível alterar e excluir uma determinada notícia como apresentado nas Figura 127 e Figura 128.

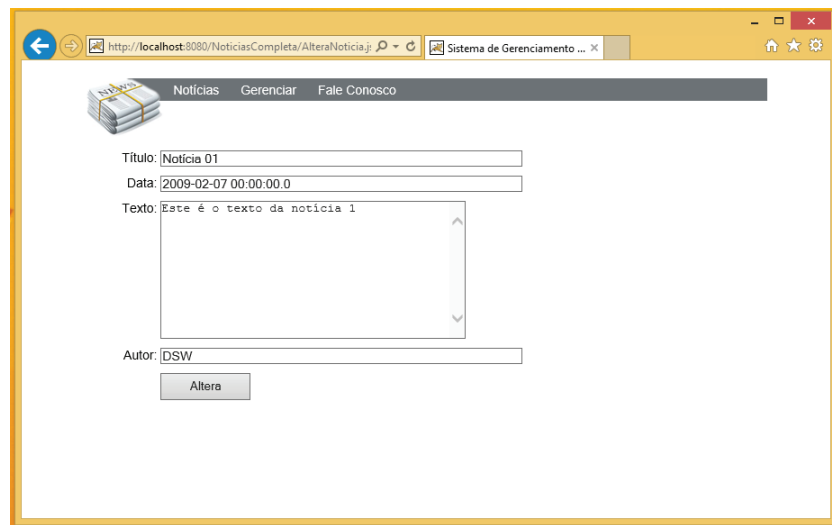


Figura 127 - Alterar notícias

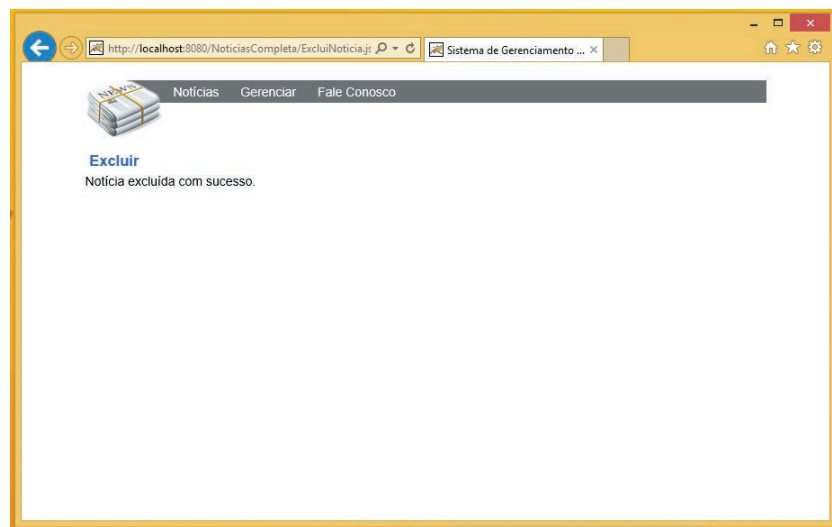


Figura 128 - Tela exibida após a exclusão de uma notícia

Além das telas de interação com o banco de dados foi criada uma tela para enviar uma mensagem. A tela só tem a interface criada, o mecanismo de email não será visto neste livro.

5.4 Considerações finais

Esta Unidade apresentou os principais conceitos associados à linguagem de programação Java para aplicações Web que usam Banco de Dados. Para tal, apresentado duas aplicações Web usando o Banco de dados MySQL.

5.5 Estudos complementares

Deitel, H. M.; Deitel, P.J. - Java Como Programar. 6ª. Edição. Editora Pearson- Prentice Hall, 2005.

Gonçalves, E. – Desenvolvendo Aplicações Web com NetBeans IDE 5.5 - Editora Ciência Moderna, 2007.

Fields, D.K.; Kolb, M.A. – Desenvolvendo na Web com JavaServer Pages – Editora Ciência Moderna, 2000.

Serson, R.R. - Certificação Java 5. Brasport Livros e Multimidia Ltda, 2006.

