

# Front-end magazine

Edição 01

 DEVMEDIA



**Padronizando código JavaScript**  
Aprenda a modularizar o seu código JavaScript com IIFE, AMD e RequireJS

**Nashorn: do conceito à prática**  
Conheça os padrões e soluções da mais nova engine JavaScript do Java 8

**Dominando a HTML5**  
Conheça os principais recursos da linguagem que revolucionou a web

# NODEJS

CRIE UMA MINI REDE SOCIAL ATRAVÉS DE DOIS TUTORIAIS COMPLETOS



## **Browser Music Player**

Implemente um player personalizado com JavaScript e HTML5

## **Menus responsivos**

Como criar um design responsivo estilo metro com HTML e CSS

# FÓRUM DEVMEDIA

O lugar perfeito para você ficar por dentro de tudo  
o que acontece nas tecnologias do mercado atual



No fórum da DevMedia você irá encontrar uma equipe disponível e altamente qualificada com consultores e colaboradores prontos para te ajudar a qualquer hora e sobre qualquer assunto. Temos as salas de Java, .NET, Delphi, Banco de Dados, Engenharia de Software, PHP, Java Script, Web Design, Automação comercial, Ruby on Rails e muito mais!

**ACESSE AGORA**  
[www.devmedia.com.br/forum](http://www.devmedia.com.br/forum)



## EXPEDIENTE

### Editor

Diogo Souza ([diagosouzac@gmail.com](mailto:diagosouzac@gmail.com))

### Consultor Técnico

Daniella Costa ([daniella.devmedia@gmail.com](mailto:daniella.devmedia@gmail.com))

### Produção

Jornalista Responsável Kaline Dolabella - JP24185

Capa e Diagramação Romulo Araujo

### Atendimento ao leitor

A DevMedia possui uma Central de Atendimento on-line, onde você pode tirar suas dúvidas sobre serviços, enviar críticas e sugestões e falar com um de nossos atendentes. Através da nossa central também é possível alterar dados cadastrais, consultar o status de assinaturas e conferir a data de envio de suas revistas. Acesse [www.devmedia.com.br/central](http://www.devmedia.com.br/central), ou se preferir entre em contato conosco através do telefone 21 3382-5038.

### Publicidade

[publicidade@devmedia.com.br](mailto:publicidade@devmedia.com.br) – 21 3382-5038

**Anúncios** – Anunciando nas publicações e nos sites do Grupo DevMedia, você divulga sua marca ou produto para mais de 100 mil desenvolvedores de todo o Brasil, em mais de 200 cidades. Solicite nossos Media Kits, com detalhes sobre preços e formatos de anúncios.

### Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique à vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site Java Magazine, entre em contato com o editor, informando o título e mini-resumo do tema que você gostaria de publicar:



DIOGO SOUZA

[diagosouzac@gmail.com](mailto:diagosouzac@gmail.com)

Analista de Sistemas Java na Indra Company e já trabalhou em empresas como Instituto Atlântico e Ebix L.A. É instrutor Android, palestrante em eventos sobre Java e o mundo mobile e consultor DevMedia. Conhecimentos e experiências em diversas linguagens e ferramentas de programação e manipulação de dados, bem como metodologias úteis no desenvolvimento de Sistemas diversificados.

## CURSOS ONLINE

A Revista Java Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.



### CONHEÇA ALGUNS DOS CURSOS:

- **Curso de noSQL (Redis) com Java**
- **Curso Básico de JDBC**
- **Java Básico: Aplicações Desktop**
- **JSF com Primefaces**
- **Conhecendo o Apache Struts**

Para mais informações :  
[www.devmedia.com.br/curso/javamagazine](http://www.devmedia.com.br/curso/javamagazine)  
(21) 3382-5038

# Sumário

Conteúdo sobre Boas Práticas

## 06 – Padronização com IIFE, AMD e RequireJS

[ William Carvalho ]

Conteúdo sobre Novidades

## 12 – Nashorn: Desenvolvendo JavaScript na JVM

[ Júlio Sampaio ]

Artigo no estilo Solução Completa

## 23 – JavaScript Blog: Criando serviço de microblog com NodeJS

[ Rafael Milléo Carrenho ]

Artigo no estilo Solução Completa

## 35 – NodeJS: Criando uma Rede Social

[ William Carvalho ]

Conteúdo sobre Boas Práticas

## 57 – Programando em HTML5

[ Henrique Poyatos ]

Artigo no estilo Solução Completa

## 68 – Interface Metro: Criando um Menu Metro com HTML e CSS

[ Joel Rodrigues ]

# SEU CARRO TEM SEGURO, SUA SAÚDE TEM SEGURO, MAS E O SEU EMPREGÓ... TÁ SEGURO??



## NÃO DEIXE JUSTAMENTE A SUA CARREIRA FICAR EM RISCO!

Manter-se atualizado com todas as novidades do mercado de desenvolvimento é obrigação de todo bom programador. Faça agora mesmo um seguro para a sua carreira. Seja um assinante MVP!

Saia do risco!



QUEM TEM ESTÁ TRANQUILO.

TENHA ACESSO A:

+DE 290 CURSOS ONLINE

09 REVISTAS MENSais

9.074 VÍDEO-AULAS

POR APENAS **69,90** MENSais



DEVMEDIA

Acesse: [www.devmedia.com.br/mvp](http://www.devmedia.com.br/mvp)

# Padronização com IIFE, AMD e RequireJS

## Como padronizar e organizar seu código JavaScript usando módulos

JavaScript vem ganhando uma espantosa força nos últimos anos e, a cada dia que passa, novas APIs e frameworks MV\* são criados por entusiastas e empresas envolvidas com a comunidade web, como foi o caso do Google com o AngularJS. Ele passou a ser o protagonista de grande parte das aplicações Web atuais.

Hoje, é possível conferir aplicações “web based” como o Google Drive, Facebook, Gmail, ou ainda é possível desenvolvermos aplicações nativas utilizando o combo HTML/CSS/JS para Windows e Firefox OS.

Com a popularização da colaboração coletiva entre desenvolvedores, houve um rápido crescimento nas pesquisas para melhorias dos engines JavaScript, como o V8 do Google, e em melhorias de código, melhores práticas e buscas por maior performance das aplicações JavaScript.

O ECMAScript, especificação na qual o JavaScript foi baseado, está cada vez mais sofisticado e trazendo novidades que vão de *syntax sugar* a melhorias críticas de performance.

O custo/tempo de desenvolvimento tende a diminuir; as aplicações web estão se tornando cada dia mais sofisticadas; o reconhecimento do JavaScript como “a linguagem da Web” faz com que os desenvolvedores falem a mesma língua.

Mas qual é a melhor maneira de escrever nosso código? Qual é o melhor padrão? Que ferramentas podemos usar como auxílio de uso desses padrões?

Linguagens dinâmicas e fracamente tipadas como o JavaScript exigem maior disciplina do desenvolvedor na hora de codificar. Diferente de linguagens estáticas e fortemente tipadas, como é o caso do Java, que precisam ser compiladas, o JavaScript pode ter seu comportamento alterado em tempo de execução. E mais ainda, em um ambiente em que inúmeros scripts podem estar sendo carregados e executados ao mesmo tempo, em concorrência.

### Fique por dentro

Conforme a tecnologia e os desenvolvedores vão amadurecendo juntos, surge a necessidade de se ter maior controle e organização do código, e a utilização de módulos veio para resolver esse problema, com ferramentas como o RequireJS que utiliza o padrão AMD (Asynchronous Module Definition). A utilização de módulos no desenvolvimento de aplicações com JavaScript é recomendada para qualquer projeto web, pois além de manter o código melhor organizado, expõe apenas o que for pertinente. Neste artigo é apresentado o desenvolvimento de um aplicativo de lista de tarefas que visa mostrar como o uso do JavaScript aliado a ferramentas como o RequireJS pode ajudar a gerenciar as dependências entre os módulos de uma aplicação.

E se um desses códigos declara um nome de variável que sobrescreve outra variável já existente? E se um plugin jQuery que está sendo executado em um laço começa a travar o browser do usuário?

Enfim, muitas dessas questões sobre concorrência e execução paralela de atividades em se tratando do universo front-end são pertinentes e neste artigo veremos como aplicar alguns destes conceitos através da construção de uma pequena aplicação de lista de tarefas, essas bem famosas no mundo web e mobile. Mas antes, vejamos alguns conceitos importantes para a construção da mesma.

### Organização do código

Em muitas linguagens de programação, é possível organizar o código através de convenções, como em Java ou frameworks como o Code Igniter para PHP ou o Django para Python.

JavaScript, por outro lado, sofre da ausência de ambos os estilos de organização. Para piorar, normalmente a camada JavaScript acaba sendo uma mistura de regra de negócio, comunicação entre o servidor e a camada de apresentação, e a manipulação da tela. Isso quer dizer que fica mais difícil separar o código em categorias lógicas reaproveitáveis.

Tomando este cenário como ponto de partida é possível observar a também necessária implantação de uma organização do código

para que o desenvolvedor não se perca dentro da implementação, tanto quanto fuja dos padrões básicos de qualidade. Pensando nisso, alguns padrões e soluções podem ser aplicados de forma a maximizar os efeitos da codificação em uma camada tão dinâmica quanto a camada cliente.

### Namespaces

Algumas ações podem ser bastante básicas e simples, mas ao mesmo tempo ajudam em muito a desenvolver um código bom. Em um projeto com um número grande de desenvolvedores, os riscos de se sobreescriver um componente importante, ou um objeto global são grandes. Foi pensando nessas possibilidades que, há alguns anos atrás, tornou-se popular a utilização de objetos literais como namespaces para evitar o conflito das propriedades.

Essa prática ajudou muito na organização do código e separação de responsabilidades, permitindo certo padrão nos verbos do CRUD e demais funcionalidades.

Este formato ainda é uma prática bastante usada, e pode funcionar bem se toda a equipe for disciplinada e compreender a importância deste isolamento. Contudo, o uso de namespaces ainda estava no escopo global window. Ou seja, ainda que os namespaces mantivessem os nomes de variáveis e funções protegidas dentro deles, eles próprios estariam contidos no escopo global, e poderiam ser acidentalmente sobreescritos ou apagados. Nesse sentido, cabe ao desenvolvedor dedicar atenção especial ao código fonte, quando estiver lidando com esse tipo de estrutura.

### Funções autoexecutáveis ou IIFE

As funções autoexecutáveis ou *Immediately-Invoked Function Expressions*, são funções que nascem, executam e morrem sem deixar muitos rastros para o escopo global. Elas basicamente não são diferentes das funções convencionais, com o diferencial de serem executadas imediatamente após serem lidas pelo interpretador.

Ao mesmo tempo, são muito úteis, entre outras coisas, para a criação de módulos, isolando atributos que não desejamos expor ao mundo exterior, dando visibilidade pública apenas aos componentes relevantes.

Essa prática é bem comum em linguagens orientadas a objeto, onde alguns métodos e atributos só fazem sentido no escopo privado, e o desenvolvedor externaliza apenas os métodos que deseja expor. Essa estratégia vem sido aplicada em diversos cenários, e tem se mostrado bastante eficiente em todos eles.

### IIFE e os boilerplate codes

Os boilerplate codes ajudam os programadores a economizar tempo na hora de começar a codificar. De forma geral, boilerplate codes são trechos de código que acabam sendo repetidos continuamente durante o tempo de desenvolvimento da aplicação. Em outras palavras, assumindo que a adoção de determinada prática ou estilo de codificar é benéfica para todos, ter o boilerplate code “na manga” faz com que não percamos tempo ao escrever algo óbvio e podemos focar no que realmente interessa. Essa prática adere ao famoso e desejado conceito de reaproveitamento de código.

Hoje é possível encontrar diversos *boilerplate codes* para IIFEs, seja para o desenvolvimento de módulos, plugins jQuery ou objetos mais complexos.

### AMD: Programando em módulos de maneira organizada

Chegamos ao ponto em que as IIFEs e os módulos se tornaram parte do nosso dia a dia. Definimos boas práticas de como organizar o código baseado na nossa experiência ou usando boilerplate codes populares.

O constante uso do JavaScript de uma forma mais avançada nos ensinou a utilizar e entender as chamadas assíncronas e funções de callback. Aprendemos que é uma boa prática posicionar nossos scripts no final do documento HTML, pois os browsers interrompem o carregamento da página para avaliar o conteúdo dos scripts.

O aperfeiçoamento dessas técnicas tem motivado iniciativas para melhorar ainda mais o desenvolvimento de aplicações web. Uma dessas iniciativas é o AMD: *Asynchronous Module Definition*.

O paradigma do AMD sugere que os módulos de uma aplicação devem ser carregados de maneira assíncrona, respeitando suas dependências e as gerenciando para que não sejam carregadas novamente caso outro módulo as solicite.

De acordo com sua especificação, uma série de métodos e regras deve ser seguida a fim de garantir o bom funcionamento da carga assíncrona e o gerenciamento das dependências entre os módulos. Ainda que não seja formalmente um padrão, o AMD tem sido largamente utilizado e tem se popularizado bastante, e é bem provável que venha a se tornar um requisito para qualquer projeto no futuro.

### RequireJS: AMD na prática

Uma das implementações que melhor reflete a utilização do AMD é a API RequireJS.

Ainda que o RequireJS não siga completamente à risca as especificações do AMD, ele representa de forma bastante concisa seu funcionamento, atendendo aos seus requisitos mais importantes: carregamento assíncrono de módulos e o gerenciamento de dependências.

Ele é composto de um arquivo js bem pequeno (cerca de 15kb minificado) e é o único arquivo que deve ser incluído no documento HTML, bastando referenciar, na mesma tag script, qual será o módulo principal da sua página.

Este módulo principal informa ao RequireJS quais são os módulos necessários para que ele funcione corretamente. Cada um desses módulos pode ter suas próprias dependências de outros módulos e assim por diante.

Se você está trabalhando em um módulo gerenciado pelo RequireJS, não precisa se preocupar se suas dependências possuem outras dependências. Se estiver trabalhando com uma aplicação utilizando a forma tradicional, você precisa conhecer todas as dependências de seus módulos e adicioná-las uma a uma com tags script no documento HTML na ordem correta de acordo com as dependências.

## Aplicativo de Lista de tarefas

Para demonstrarmos a eficácia do RequireJS, vamos implementar a famosa To do List. Essa aplicação simples utiliza AngularJS na camada MVC, Twitter Bootstrap para a identidade visual e um pequeno wrapper para trabalhar com IndexedDB de maneira mais simples.

No decorrer do desenvolvimento da aplicação, diversos módulos serão criados, cada qual com suas dependências, todas gerenciadas pelo RequireJS.

A estrutura de pastas e arquivos da aplicação será igual à apresentada na **Listagem 1**.

### Listagem 1. Estrutura de diretórios do projeto

```
css
  bootstrap.min.css
  bootstrap-theme.min.css
fonts
index.html
js
  controllers
    task.js
  localdb
    db.js
  main.js
  todo.js
lib
  angular.min.js
  angular-route.min.js
  bootstrap.min.js
  jquery.min.js
  require.js
  zondb-min.js
partials
  home.html
```

Alguns arquivos do projeto foram ocultados para que possamos nos concentrar nos diretórios mais relevantes do mesmo. O diretório lib contém todas as bibliotecas das quais o aplicativo utiliza, e o diretório js possui subdiretórios com os controllers, arquivo principal e a biblioteca de utilização do IndexedDB.

No mesmo nível do diretório js está o diretório partials, com os templates do AngularJS.

Conforme explicado anteriormente, nossa aplicação possui apenas uma importação de arquivos JS contendo a chamada para o require.js e um atributo data-main apontando para o arquivo principal da aplicação, conforme o conteúdo do index.html exibido na **Listagem 2**.

Quando o RequireJS é incluído, procurará pelo atributo data-main e tentará carregar esse arquivo e suas dependências.

É também nesse arquivo, no nosso caso, js/main.js (a extensão .js é omitida pois o RequireJS assume que todos os módulos são arquivos JavaScript), onde estão as configurações do RequireJS, tais como bibliotecas, caminhos e mapeamentos como mostrado na **Listagem 3**.

### Listagem 2. Página HTML e o include único do RequireJS

```
<!doctype html>
<html>
<head>
  <title>Welcome to the new project</title>
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css"/>
  <link rel="stylesheet" type="text/css" href="css/bootstrap-theme.min.css"/>
</head>
<body ng-app="Todo">

  <div class="container">
    <div class="row">
      <div class="col-md-2"></div>
      <div class="col-md-8">
        <h1>Meu TODO List</h1>
        <div ng-view></div>
      </div>
      <div class="col-md-2"></div>
    </div>
    <script type="text/JavaScript" data-main="js/main" src="lib/require.js"></script>
  </body>
</html>
```

### Listagem 3. Configuração do RequireJS no arquivo main

```
window.name = "NG_DEFER_BOOTSTRAP!";

requirejs.config({
  baseUrl: 'js',
  paths: {
    'bootstrap': '../lib/bootstrap.min',
    'angular': '../lib/angular.min',
    'angular-route': '../lib/angular-route.min',
    'zondb': '../lib/zondb'
  },
  shim: {
    'angular': {
      exports: 'angular'
    },
    'angular-route': {
      deps: ['angular']
    },
    'zondb': {
      exports: 'zonDB'
    }
  }
});

require(['todo'], function(todo) {
  angular.element().ready(function() {
    angular.resumeBootstrap();
  });
});
```

A função **requirejs.config** recebe um objeto como argumento com as configurações da aplicação, e como o RequireJS deve lidar com as dependências.

O atributo **basePath** diz a partir de qual diretório os módulos devem ser carregados sem a necessidade de especificar um caminho. Ainda é possível utilizar caminhos relativos “abaixo” do diretório base, porém, este deve ser um caminho de fácil acesso aos arquivos da aplicação.

O atributo **paths** mapeia bibliotecas que precisamos utilizar, por qual nome elas serão reconhecidas e em qual diretório elas se encontram.

É possível, por exemplo, mapear mais de uma versão de uma mesma biblioteca com nomes diferentes, já que o RequireJS irá permitir apenas um nome de módulo exclusivo dentro de seu contexto. Isso garante que não haja conflitos de nomes, evitando a perda de módulos que viriam a ser sobreescritos, e cujo problema seria muito difícil de detectar posteriormente.

O atributo **shim**, que em português quer dizer calço, serve para incluir bibliotecas de contexto global dentro do contexto do AMD.

JQuery, AngularJS e outros frameworks residem no contexto global window, o que vai contra o paradigma do AMD. Para resolver esse problema, criamos esses calços, que importam esses frameworks imitando o comportamento do AMD.

Após a configuração da aplicação, utilizamos a função **require** para executar o main.js, carregando todas as suas dependências. Esta função recebe dois argumentos: o primeiro é um array de strings opcional com a lista de módulos do qual o main.js depende, e o segundo argumento é uma função de callback que executará efetivamente o nosso programa principal.

À medida que vamos incluindo módulos no array de strings, cada módulo carregado será um argumento passado para a função de callback.

No caso de nosso main.js está carregando o módulo todo.js (lembre-se que o RequireJS assume que todos os módulos são arquivos js, e portanto não devemos explicitar a extensão na importação dos mesmos).

Vamos então dar uma olhada no módulo todo.js, conforme mostrado na **Listagem 4**.

#### Listagem 4. Módulo todo.js

```
define(['angular', 'controllers/task', 'angular-route'],
function(ng, Task) {
    var Todo = ng.module('Todo', ['ngRoute']);
    Todo.controller('Task', Task);

    Todo.config(function($routeProvider, $locationProvider) {
        $routeProvider.when('/', {
            templateUrl: 'partials/home.html',
            controller: Task
        });
    });

    return Todo;
});
```

O módulo todo.js cria um módulo do AngularJS chamado **Todo** e o retorna ao final da função define, do RequireJS. Notem que a função define funciona da mesma maneira que a função require, porém seu intuito é exportar um módulo, enquanto que a função require tem a finalidade de executar instruções, ao invés de exportá-las.

Por se tratar de um módulo do AngularJS, ele depende da biblioteca do Angular e ngRoute, e também carrega uma controller chamada de Task.

Todas essas dependências estão sendo carregadas no primeiro argumento da função require, e os objetos relevantes (note que não especificamos uma referência para o ngRoute, pois não o utilizaremos explicitamente) são passados como argumento para a função de callback, para que este possa ser usado na definição do módulo Todo.

Tanto angular quanto angular-route são módulos que configuramos como sendo bibliotecas que nossa aplicação depende, porém o módulo task foi desenvolvido exclusivamente para a aplicação.

Repare que, por estar em outro diretório, seu caminho precisa ser detalhado na lista de módulos a partir do **basePath** que definimos anteriormente.

O conteúdo do arquivo controllers/task.js é exibido na **Listagem 5**.

Este módulo é um pouco mais extenso, pois possui a interface de comunicação com a biblioteca de manipulação de dados do IndexedDB e todos os CRUDs, bem como a cola entre os dados e a camada de apresentação através do scope.

O módulo Task depende essencialmente das bibliotecas responsáveis por essas ações, ou seja, o angular e o wrapper do IndexedDB.

**Não perca tempo  
reinventando a roda!**

## COBREBEMX

**Componente completo para sua  
Cobrança por Boleto Bancário  
e Débito em Conta Corrente**

Mais de 40 exemplos  
em diversas linguagens  
de programação

Geração e leitura de arquivos  
(remessa e retorno) nos padrões  
**FEBRABAN e CNAB**

Testes e Downloads  
gratuitos em nosso site

ACESSE E CONHEÇA O COMPONENTE EM:  
**WWW.COBREBEM.COM**

## Listagem 5. Módulo controllers/task.js

```
define(['angular', './localdb/db'], function(ng, db) {
  function Task($scope, $rootScope, $timeout) {
    $scope.title = "My tasks";
    $scope.taskName = null;
    $scope.tasks = null;
    $scope.taskId = null;

    function getRowIndex(taskId) {
      var i=0,
          len = $scope.tasks.length;

      for(i; i < len; i++) {
        if($scope.tasks[i].id == taskId) {
          return i;
        }
      }
    }

    function loadTasks() {
      db.query('tasks', function(res) {
        $scope.tasks = res;
        $scope.$digest();
      });
    }

    $scope.saveTask = function() {
      var task = {name: $scope.taskName};

      if($scope.taskId) {
        task.id = $scope.taskId;
        updateTask(task);
        return;
      }

      db.addRow('tasks', task, function(data) {
        task.id = data;
        $scope.tasks.push(task);
        $scope.taskName = null;
        $scope.$digest();
      });
    };

    $scope.removeTask = function(taskId) {
      $scope.tasks.forEach(function(obj, idx) {
        if(obj.id == taskId) {
          db.deleteRow('tasks', taskId, function() {
            $scope.tasks.splice(idx, 1);
            $scope.$digest();
          });
        }
      });
    };

    $scope.editTask = function(taskId) {
      $scope.taskId = taskId;
      var task = $scope.tasks.filter(function(task, idx) {
        if(task.id == taskId) return task;
      })[0];

      $scope.taskName = task.name;
    };

    loadTasks();
  }

  return Task;
});
```

Note que no decorrer da construção da aplicação, muitas dependências de módulos se repetem. Isso é resolvido internamente pelo RequireJS, ou seja, ele faz o controle de módulos que já foram carregados, evitando que sejam carregados novamente.

Se algum dos módulos a ser carregado possui outras dependências, isso também é resolvido pelo RequireJS que fará o download assíncrono e sob demanda dos arquivos somente quando for necessário.

No final, se você fizer o deploy dessa aplicação em qualquer servidor web, terá um resultado semelhante ao mostrado na **Figura 1**.

A aplicação basicamente se encarrega de adicionar tarefas, editá-las e removê-las, ações básicas de um CRUD, porém usando o que há de mais organizado no que se refere a JavaScript em conjunto com o RequireJS.

## Otimização dos módulos

Além disso, é importante mencionar que o RequireJS conta com uma ferramenta otimizadora chamada **r.js**.

## Meu TODO List

Task name:		<input type="button" value="Save"/>
<b>Id</b>	<b>Task description</b>	
2	do something else	<input type="button" value="x"/> <input type="button" value="edit"/>
3	Buy beer	<input type="button" value="x"/> <input type="button" value="edit"/>

Figura 1. Exemplo da tela da aplicação

Essa ferramenta precisa ser baixada separadamente e depende de outras ferramentas para ser executada, como por exemplo, **NodeJS**.

Com ela é possível, por exemplo, unir módulos interdependentes e criar um único arquivo minificado. Por exemplo, suponha que você possua um módulo usuário, que foi separado nas camadas **view**, de apresentação, e **model**, com as ações de CRUD.

Esses dois submódulos só fazem sentido para o seu módulo principal usuário. Logo, ao serem preparados para o ambiente de produção, podem seguramente ser unidos em um único arquivo e minificados.

O r.js é capaz de realizar esse merge sem prejudicar as dependências que você criou na fase de desenvolvimento. Funciona mais ou menos como se o r.js compilasse sua aplicação, reduzindo a quantidade de arquivos e seus tamanhos, e, dependendo da forma utilizada para minificar os arquivos, pode até otimizar o seu código.

Isso constitui mais uma ferramenta à mão dos desenvolvedores que facilitariam significativamente o desenvolvimento e manutenção do código, assim como implicariam em melhorias de performance e otimização.

Muito se tem discutido e implementado sobre as funções autoexecutáveis IIFEs e o aparecimento de módulos, tanto que se propagou por todo lado, o que nos deixou mais próximos de o que poderia vir a ser um padrão de desenvolvimento.

A experiência adquirida pelos desenvolvedores e o aperfeiçoamento dos módulos e utilização das IIFEs levou à criação e expansão do AMD, que reforça a utilização de módulos que possam ser carregados de forma assíncrona, e que sejam gerenciados de maneira inteligente.

A responsabilidade de manter o código limpo e organizado pertence, obviamente, ao desenvolvedor e o AMD pode fazer isso pelo mesmo. Esse artigo mostrou através do desenvolvimento de uma pequena aplicação, que as pequenas partes podem de fato fazer uma grande diferença no final.

#### Autor



Willian Carvalho

[o.chams@gmail.com](mailto:o.chams@gmail.com)

Programador desde 2000, trabalha com desenvolvimento para web, e já passou por ASP, PHP e principalmente Java. Formado em Tecnologia da Informação pela FASP. Atualmente é frontend tech lead na TOTVS onde trabalha com JavaScript e NodeJS.



#### Links:

##### **Site do RequireJS**

<http://requirejs.org/>

##### **Documentação da API do RequireJS**

<http://requirejs.org/docs/api.html>

##### **AMD (Asynchronous Module Definition)**

<https://github.com/amdjs/amdjs-api/wiki/AMD>

# DÊ UM SALTO EM CONHECIMENTO!



Acesse o maior  
portal para  
desenvolvedores  
da América  
Latina!

20  
mil  
posts

430  
mil  
cadastrados

10  
milhões de  
page-views  
por mês

# Nashorn: Desenvolvendo JavaScript na JVM

Conheça todas as principais features do sucessor do Rhino para desenvolver JavaScript na JVM

Você provavelmente já leu ou ouviu falar sobre algumas das principais características da última versão do Java SE 8 e JDK 8, tais como:

- Projeto Lambda (JSR 335), o que torna mais fácil escrever código para processadores multi-core,
- Um conjunto de perfis compactos, que permitem implementações do Java SE 8 de baixa escalabilidade com facilidade,
- Uma nova API de data e hora (JSR 310),
- E a remoção da “geração permanente” do JVM HotSpot.

Todos esses recursos incorporaram às novas versões da linguagem grandes esforços de engenharia, implementação e praticidade no que concerne ao desenvolvimento de softwares como até então havia sendo feito.

O projeto **Nashorn**, outra característica fundamental do JDK 8, não é uma exceção: ele introduz uma nova implementação mais leve e de alto desempenho do JavaScript e integra-a ao JDK.

Mas antes que você comece a se perguntar sobre toda essa confusão de ter duas expressões até então tidas como bem distintas e diferentemente localizadas: JavaScript (no lado cliente das aplicações) e Java (no lado servidor), voltemos um pouco a linha do tempo para entender como tudo começou e qual o objetivo de se usar as duas linguagens em conjunto.

## JavaScript na JVM

No princípio tínhamos a *JSR 223: Scripting for the Java platform*. Essa JSR foi a principal responsável pela criação do pacote que viria a ser o precursor de toda inclusão JavaScript dentro do Java: o `javax.script` (também podendo ser chamado de API JavaScript). Essa API `javax.script` fornece uma linguagem de script independente para usar tais linguagens diretamente do Java. Ao mesmo tempo essa API somente foi incluída pela primeira vez na versão da plataforma Java SE 6, tendo como

## Fique por dentro

O desenvolvimento de aplicações web tem se mostrado cada vez mais forte no mundo das linguagens de programação front-end. Ao mesmo tempo, algumas linguagens server side não perdem a robustez e preferência entre os mesmos, dentre elas estão Java e JavaScript, semelhantes no nome, mas bem diferentes funcionalmente. O Nashorn é o mais novo projeto incluído na última versão do Java, em substituição ao Mozilla Rhino que renova o conceito de comunicação entre as duas linguagens citadas: é possível chamar Java a partir de JavaScript e vice versa.

Neste artigo veremos de forma prática desde os conceitos mais básicos de instalação e configuração, até a comunicação do Nashorn com APIs como o JavaFX e Avatar.js, de forma a passar uma ideia do poder e capacidade dessa tecnologia.

empacotamento principal o mecanismo de scripts baseado no Mozilla Rhino para JavaScript.

Mozilla Rhino é uma implementação open source do JavaScript baseada na linguagem de programação Java. Ele leva o nome de “rinoceronte” em alusão a um popular livro de desenvolvimento em JavaScript. Ao longo dos anos, no entanto, seu desempenho caiu muito frente a outros engines JavaScript. Para melhorar o desempenho, o Rhino precisaria ser reescrito para explorar plenamente as capacidades da plataforma Java moderna.

Mas o porquê então de usar JavaScript junto com Java? Bem, o principal benefício de se fazer isso seria solucionar um problema extremamente comum no mundo das aplicações web: a validação das regras de negócio.

Imagine a seguinte situação: você está desenvolvendo um aplicativo baseado na web para uma finalidade qualquer, e aí você se depara com o comum desafio de escrever código para efetuar as validações da lógica de negócios. A maioria dos frameworks web apoia a noção de validações em nível de campo para assegurar, por exemplo, que um campo de texto que recebe o valor da idade de uma pessoa permita apenas números inteiros positivos entre 0 e 120, ou que um endereço

de e-mail siga o padrão de `usuario@dominio.dominiotoplevel`. Mas o que frameworks web têm dificuldade em validar são as suas regras de negócio.

Para solucionar esse tipo de situação as ações comuns de muitos desenvolvedores geralmente são:

- Cliente: não é interessante que os usuários tenham que preencher um formulário inteiro antes de notificar-lhes que um campo é inválido, então no lado do cliente, usa-se JavaScript para validar os valores de campo individuais;
- Servidor: todas as alegações finais para o servidor devem ser validadas usando um engine de regras; este pode ser feito em Java mesmo, ou em linguagens semelhantes como Groovy ou Scala.

Um dos objetivos de qualquer aplicação web ante um negócio específico é fornecer aos usuários um feedback imediato sobre os dados que estão entrando, assegurando também a integridade das regras de negócio no servidor. Mas isso cria um desafio, ou seja, temos de escrever muitas das regras de negócio duas vezes: uma vez no lado do cliente e outra no do servidor. O desafio é ainda agravado pela manutenção. Por exemplo, usando a mesma situação apresentada, imagine o que aconteceria se o negócio decidisse que a faixa de idade válida passará a ser entre 4 e 130? A resposta é que teríamos de atualizar o JavaScript na página, bem como as regras no servidor back-end.

Diante desse cenário é onde entra o Nashorn (assim como o Rhino, antigamente), pois ele permite a execução de uma regra só em ambas as camadas.

Adicionalmente, é possível encontrar JavaScript no lado servidor em outras situações. Por exemplo, o Node.js é usado para construir servidores leves e rápidos baseado no engine V8 JavaScript do Google Chrome. Engines JavaScript em navegadores da Web têm acesso ao modelo de objeto de documento HTML (DOM) e podem manipular elementos HTML através do DOM. Dado que diferentes navegadores Web têm diferentes DOM e engines de JavaScript, frameworks como jQuery tentam esconder os detalhes de implementação do programador.

## Visão geral do Nashorn

Nashorn, pronuncia-se “nass-horn”, em alemão significa “rincôrante”, e é um dos nomes de animais para um contratorpedeiro de tanque alemão usado na Segunda Guerra Mundial. É também o nome do substituto - introduzido com o novo Java 8 - para a velha e lenta engine Rhino JavaScript, conforme falamos na seção anterior. Ambos Rhino e Nashorn são implementações da linguagem JavaScript escritos para executar na máquina virtual Java.

Ele nasceu da iniciativa de um projeto no OpenJDK Community, em 2012, e por ter o código que compõe a implementação todo open source, atraiu adotantes e melhorou rapidamente, a ponto de passar por todos os testes de conformidade ECMAScript. Diante disso, a tecnologia foi escolhida para compor a próxima versão do JDK 8 e evoluiu para uma das principais características desse lançamento.

O JavaScript, também, tem evoluído nos últimos anos, e é possível observar o seu uso fora do nicho original, dentro de navegadores web. Isso pode fazer uma implementação totalmente compatível, com alto desempenho de JavaScript na JVM cada vez mais atraente para os desenvolvedores Java e JavaScript.

JavaScript pode ter Java como parte do seu nome, mas as duas línguas são muito diferentes em espírito e design, bem como em suas implementações. No entanto, uma forma de implementar um interpretador de JavaScript é compilar JavaScript em bytecode, que é o que o Rhino e o Nashorn foram projetados para fazer.

Nashorn, e Rhino, antes disso, explicitamente não suportam o DOM do navegador. Implementados na JVM, eles normalmente são chamados para scripting do usuário final em aplicações Java. Nashorn e Rhino podem ser incorporados em programas Java e usados como linhas de comando shell. Claro, o esforço adicional necessário quando você está criando scripts Java a partir do JavaScript preenche a lacuna entre os dados e tipos de incompatibilidade das duas linguagens.

### Recursos e Melhorias

Jim Laskey, consultor membro da equipe técnica da Oracle, descreveu os objetivos do Nashorn da seguinte forma:

- Será baseado na especificação da linguagem ECMAScript-262 Edition 5.1 e deve passar por todos os testes de conformidade da ECMAScript-262.
- Apoiará a API `javax.script` (JSR 223).
- Será concedido apoio para invocar código Java a partir de JavaScript e Java para invocar código JavaScript. Isso inclui mapeamento direto para JavaBeans.
- Irá definir uma nova ferramenta de linha de comando, JJS, para avaliar o código JavaScript em scripts shell.
- O desempenho e uso de memória de aplicações Nashorn devem ser significativamente melhores do que do Rhino.
- Não vai expor quaisquer riscos de segurança adicionais.
- As bibliotecas fornecidas devem funcionar corretamente sob localização.
- Mensagens de erro e documentação serão internacionalizadas.

Laskey também limita explicitamente o escopo do projeto com alguns “não objetivos”:

- Só apoiará ECMAScript-262 Edição 5.1. Não vai apoiar quaisquer características da Edição 6 ou quaisquer recursos não padronizados fornecidos por outras implementações de JavaScript.
- Não irá incluir um plug-in API para browser.
- Não incluirá suporte para DOM/CSS ou quaisquer bibliotecas relacionadas (tais como jQuery, Prototype, ou Dojo).
- Não incluirá suporte a debugging direto.

Então, o que significa dizer que será baseado na edição 5.1 do ECMAScript-262? O diferencial aqui é que o Rhino foi baseado numa especificação mais antiga, na Edição 3.

A falta de suporte de depuração no Nashorn é um passo para trás a partir do Rhino, que tem seu próprio depurador JavaScript.

No entanto, você vai encontrar soluções alternativas para essa omissão deliberada em pelo menos duas IDEs populares.

## Observação

Nashorn é a única engine JavaScript incluída no JDK. No entanto, você pode usar qualquer mecanismo de script compatível com a JSR 223, ou implementar o seu próprio.

## Interoperabilidade

Em vez de ser apenas mais uma engine JavaScript, o Nashorn fornece interoperabilidade entre os mundos Java e JavaScript. Isso significa que seu código Java pode chamar o código JavaScript, e vice-versa.

O Nashorn fornece objetos globais para acessar e instanciar classes Java a partir de JavaScript. Seus membros podem ser acessados usando a familiar notação ‘.’ como no Java. Getters e setters nos JavaBeans são expostos como propriedades do JavaScript equivalentes. Arrays Java podem ser instanciados e acessados a partir do JavaScript, enquanto arrays de JavaScript podem ser convertidos em arrays Java comuns, quando necessário. Você pode usar “for” e “for each” para percorrer os dois tipos de arrays. Strings e números são tratados de forma transparente, interpretando-os como instâncias das classes Java correspondentes, dependendo da operação realizada. Finalmente, as coleções são interpretadas como arrays também.

Além disso, você pode estender as classes Java, fornecendo uma função de JavaScript que implementa um novo método. O Nashorn pode estender automaticamente métodos de classes simples e abstratas se você fornecer a implementação do novo método no construtor. Isto leva a um código muito compacto para action listeners, por exemplo.

## Shell Scripting

O Nashorn vem com uma série de pequenas extensões para torná-lo mais fácil de usar JavaScript no shell. Elas são ativadas passando a flag `-scripting` para a aplicação `jjs`. Mas se o seu script começar com os caracteres (#!) e o caminho direto para a aplicação `jjs`, você não precisa passar a mesma.

Você pode usar interpolação de string `(${var})` para usar os valores das variáveis ou expressões para a construção de strings dentro de aspas duplas. Você também pode usar um here document (`heredoc`) para especificar strings que preservam as quebras de linha e recuo. As variáveis de ambiente, os argumentos de linha de comando, o output e strings de erro estão disponíveis como objetos globais, bem como o código de saída do script e uma função global para executar comandos.

Além disso, o Nashorn fornece várias funções internas para sair do script, imprimir e ler as linhas de saída padrão e de entrada, ler arquivos, scripts de carga, e as propriedades de vinculação de objetos.

## Performance e bytecodes

Por detrás dos panos, o Nashorn usa a instrução JVM `invoke-dynamic` para implementar todas as suas invocações. Esse é um

componente importante na melhoria comparativa do desempenho e uso de memória sobre o Mozilla Rhino. Ao contrário do Java ou Scala, cujos compiladores são persistentes (ou seja, geram arquivos de classe/jar para o disco), o Nashorn compila tudo na memória e passa o bytecode para a JVM diretamente.

Uma vez que o JavaScript não tem um formato de bytecode “nativo”, o código fonte JavaScript é analisado em primeiro lugar para a construção de uma representação imediata (AST/IR). A AST/IR é então reduzida para algo mais próximo do bytecode JVM pela transformação de controles, redução de expressões para operações primitivas, e a simplificação das chamadas, a ser traduzido de forma eficiente com as instruções da JVM, e seguramente carregado na mesma. O código gerado e o histórico de chamadas são armazenados em cache pelo linker, para fazer lookup e invocação mais rapidamente em relinks sucessivos - JavaScript sendo uma linguagem dinâmica, o código real que precisa ser executado pela JVM para uma função que está sendo chamado em um determinado ponto no código pode mudar ao longo do tempo, e precisa ser recompilado e reconectado (relinked). O Nashorn cuida de tudo implicitamente através de bibliotecas auxiliares de alta qualidade provindas de terceiros.

O ganho em performance é tamanho que uma análise mais a fundo no desempenho comparado entre as tecnologias do Rhino e o próprio Nashorn, pode ser averiguado através da **Figura 1**.

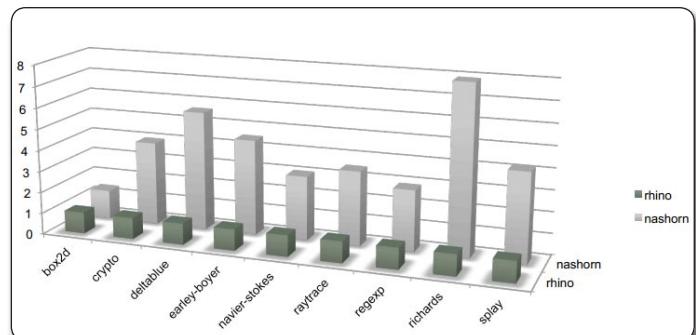


Figura 1. Análise de desempenho entre as engines JavaScript Nashorn e Rhino

## Command-line: instalando o JSS e o jrunscript

Uma vez que você instalar um JDK 8, ou construir sua própria instalação a partir do código-fonte do OpenJDK, você será capaz de usar a engine Nashorn através da API `javax.script` e a ferramenta `jrunscript`, assim como usaria com o Rhino. Além disso, o Nashorn vem com um novo aplicativo de linha de comando chamado `jjs`, que está localizado no diretório bin da instalação do seu JDK 8. A ferramenta `jjs` pode ser usada para executar e avaliar programas de JavaScript diretamente, tornando-o fácil de usar JavaScript para escrever shell scripts, código de protótipo na linha de comando, e até mesmo escrever aplicações JavaFX em JavaScript.

A primeira coisa que precisa fazer é verificar se o seu Sistema Operacional reconhece o JDK 8 como padrão na sua máquina, para tal acesse o prompt de comando cmd e digite o comando:

```
java -version
```

### Nota

Neste artigo usaremos como padrão as configurações relacionadas ao Sistema Operacional Windows 7. Mas as mesmas podem ser aplicadas em semelhança aos demais SOs e respectivas versões.

Se o resultado for equivalente ao exibido na **Listagem 1**, então você já tem tudo que precisa para trabalhar com o Nashorn. Caso contrário, instale a versão correta do JDK e refaça estes passos. Desconsidere os “updates” que são os números que vem após o numero da versão em si, 1.8.0\_05, por exemplo.

#### Listagem 1. Resultado de comando de execução do Java version

```
java version "1.8.0"
Java(TM) SE Runtime Environment (build 1.8.0-b132)
Java HotSpot(TM) 64-Bit Server VM (build 25.0-b70,
mixed mode)
```

Após isso você estará apto a executar por default a ferramenta jrunscript, que já vem acoplada à instalação comum do JDK. Digite o seguinte comando a seguir para executar o bat do jrunscript e testar um alert JavaScript, ao mesmo tempo:

```
jrunscript
js> alert("Alo Mundo, Devmedia!");
```

O resultado será semelhante ao representado pela **Figura 2**.

Esse resultado serve para explicitar e lembrar ao leitor acerca da importância de considerar que a execução de scripts via geração de bytecodes não leva em conta a existência de um DOM ou sequer objetos de browser. A função de alerta “alert()” faz uso desses recursos ao solicitar ao browser exibir graficamente uma mensagem passada por parâmetro.

Note também que o estilo de mensagens de erro no mesmo se equipara ao do Java, uma vez que estamos executando código bytecode no fim das contas. A presença da classe EcmaError também reforça a ideia de padronização JavaScript.

No entanto, se tentarmos a execução de um código diferente, porém reconhecido por ambas as linguagens, como o método

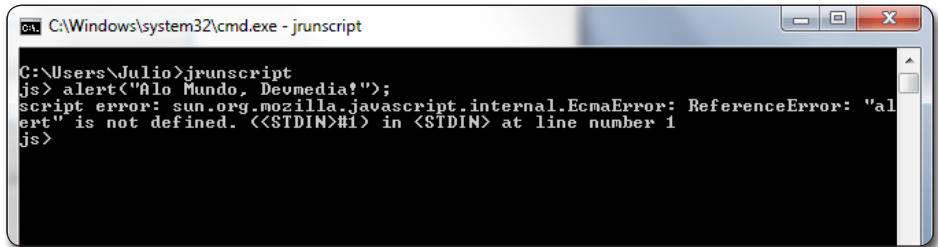


Figura 2. Resultado da execução de um alert no jrunscript

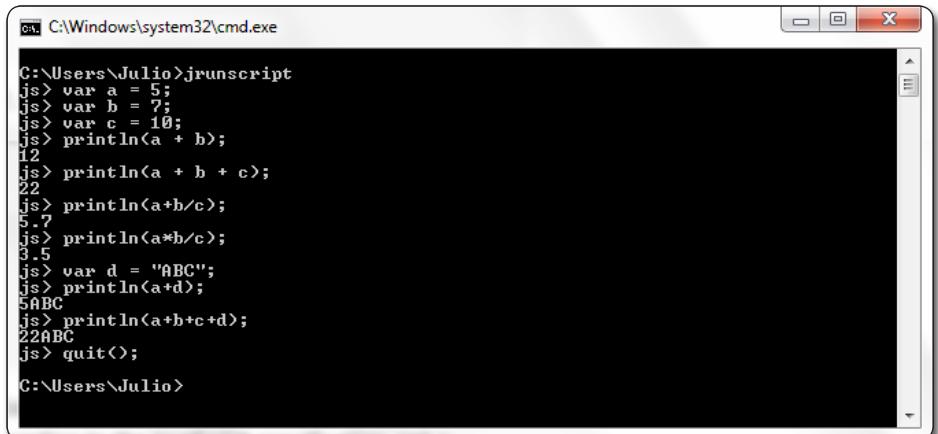


Figura 3. Resultado da execução da Listagem 2

print, então teríamos um resultado com sucesso:

```
js> print("Alo Mundo, Devmedia!");
Alo Mundo, Devmedia!
```

Outros tipos de operações básicas como a associação de valores do tipo literal ou a adição de valores numéricos, bem como demais operações aritméticas são perfeitamente possíveis de serem executados através do jrunscript. Veja o caso representado pela **Listagem 2**. Nele você pode conferir a execução de todos esses casos e o resultado final pode ser verificado na **Figura 3**.

Ainda no contexto do exemplo anterior, é possível trabalhar observando um dos conceitos mais complexos da implementação do Nashorn: a forte diferença de tipagens entre as duas linguagens. Através da função typeof, presente em ambas as linguagens, é possível observar o que o compilador considera como tipo básico para o valor em questão:

```
js> print(typeof(a+d));
string
```

#### Listagem 2. Operações aritméticas e de associação usando jrunscript

```
var a = 5;
var b = 7;
var c = 10;
println(a + b);
println(a + b + c);
println(a+b/c);
println(a*b/c);
var d = "ABC";
println(a+d);
println(a+b+c+d);
quit();
```

Isso é nada mais que um elegante efeito colateral da tipificação fraca e sobrecarga do operador “+” em JavaScript. É o comportamento correto de acordo com a especificação JavaScript, e não um bug.

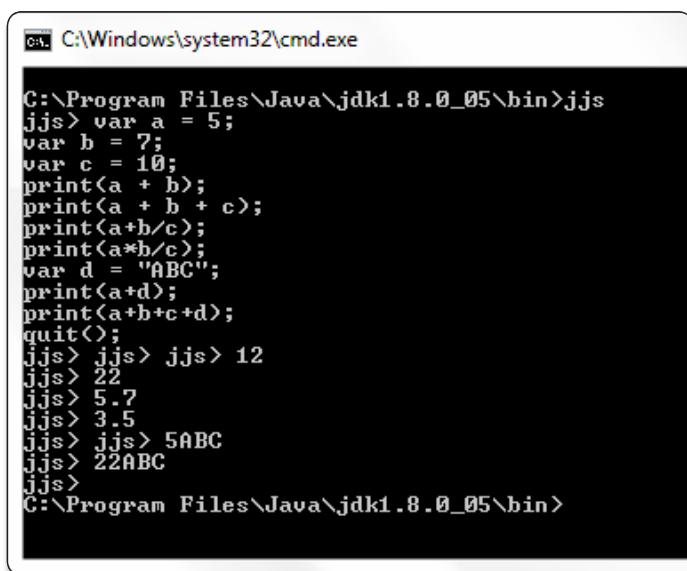
O Nashorn suporta o caractere “#” como um marcador principal de linha de comentário, assim tanto jjs como jrunscript podem ser usados em scripts escritos em JavaScript. Se estiver usando Mac ou Linux, você terá que marcar o arquivo JavaScript como executable com o utilitário chmod para torná-lo executável.

Mas e o jss, onde fica? Se você executar jjs na linha de comando do seu prompt,

provavelmente receberá a seguinte mensagem de erro:

“jjs’ não é reconhecido como um comando interno ou externo, um programa operável ou um arquivo em lotes.”

Isso acontece porque o jjs não é configurado no diretório de binários padrão do Java quando da sua instalação. Para tal você precisaria acessá-lo diretamente. Geralmente, o caminho padrão de instalação (para Windows) é: C:\Program Files\Java\jdk1.8\bin, e lá você irá encontrar um arquivo bat chamado jjs.bat. Basta executá-lo ou acessar o mesmo através do seu próprio prompt. Veja na **Figura 4** o resultado da execução do mesmo script da **Listagem 2** para efeito de comparação entre ambas as ferramentas.



```
C:\Windows\system32\cmd.exe
C:\Program Files\Java\jdk1.8.0_05\bin>jjs
jjs> var a = 5;
var b = 7;
var c = 10;
print(a + b);
print(a + b + c);
print(a+b/c);
print(a*b/c);
var d = "ABC";
print(a+d);
print(a+b+c+d);
quit();
jjs> jjs> jjs> 12
jjs> 22
jjs> 5.7
jjs> 3.5
jjs> jjs> 5ABC
jjs> 22ABC
jjs>
C:\Program Files\Java\jdk1.8.0_05\bin>
```

Figura 4. Comparação entre a execução no jjs e jrunscript

Observe que para esta execução retiramos a presença do método println(), substituindo-o pelo print(), uma vez que o jjs não o reconhece como função válida.

Adicionalmente, você pode habilitar um modo de scripting no jjs que não existe no jrunscript. No modo de criação de scripts, expressões dentro de aspas simples são passadas para a camada externa para a avaliação (**Listagem 3**).

**Listagem 3.** Execução do jjs em modo scripting

```
jjs -scripting
jjs> print ('alo');
Applications
Applications (Parallels)
Creative Cloud Files
Desktop
...
work
jjs>
```

## Chamando JavaScript a partir do Java

Para um primeiro momento desenvolveremos um projeto de teste, que nos auxiliará com a composição das listagens e respectivos testes no código, tanto Java quanto em JavaScript. Para tanto, você precisará da IDE Eclipse, em sua versão clássica, qualquer versão que já aceite a versão 8 do Java.

Após instalar o Eclipse, crie um novo projeto Java simples, aqui o chamaremos de “nashorn-devmedia”, e nele colocaremos todo o fonte que se refira aos testes com Nashorn que fizemos daqui pra frente.

Crie também uma classe e nomeie-a Teste, com um método main implementado para que possamos efetuar a chamada aos códigos JavaScript, tal como na **Listagem 4**.

**Listagem 4.** Código da classe Teste para executar os scripts JavaScript

```
package br.edu.devmedia.nashorn.alomundo;
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;

public class Teste {

    public static void main(String[] args) {
        new Teste().executarViaArquivoJS();
        //new Teste().executarViaCodigoInline();
    }

    private void executarViaArquivoJS() {
        try {
            ScriptEngineManager factory = new ScriptEngineManager();
            ScriptEngine engine = factory.getEngineByName("nashorn");
            engine.eval("load(\"src/js_java_exemplo.js\")");
        } catch (ScriptException ex) {
            ex.printStackTrace();
        }
    }

    private void executarViaCodigoInline() {
        try {
            ScriptEngineManager factory = new ScriptEngineManager();
            ScriptEngine engine = factory.getEngineByName("nashorn");
            engine.eval("function ola() {\n    var ola = 'DEVMEDIA'.toLowerCase();\n    \n    iterar();\n    print('Ola Mundo' + ola + '!');\n}\n\nfunction iterar()\n{\n    var cont = 1;\n    for (var i = 0, max = 5; i < max; i++) {\n        cont++;\n        \n        print('Valor da var cont: ' + cont);\n    }\n}\n\nola();");
        } catch (ScriptException ex) {
            ex.printStackTrace();
        }
    }
}
```

Para chamar um script JavaScript a partir do Java via Nashorn você basicamente precisa implementar uma nova instância de ScriptEngineManager e usá-la para carregar o script pelo nome, através do método getEngineByName(). Este método existe para ajudar o Java a identificar que tipo de engine de scripts está prestes a ser usado, diante da possível existência de outros carregados no classpath das aplicações Java.

Em seguida, é possível verificar a chamada direta ao método “eval()” que se encarrega de avaliar a expressão JavaScript recebida em forma de parâmetro String e executar a mesma. A função desse método é basicamente a de executar o código JavaScript recebido como se simulasse um interpretador JavaScript de um browser comum.

Consequentemente, o entendimento do código da **Listagem 4** não poderá ser totalmente compreendido sem a criação dos arquivos JavaScript correspondentes. Note que o método main faz a chamada a dois métodos internos privados que se caracterizam por demonstrar os dois possíveis meios de se carregar código JavaScript a partir do Java: via código inline, ou através da importação dos próprios arquivos JavaScript diretamente no código Java.

Dentro da pasta “src” do seu projeto Java, crie um novo arquivo JavaScript selecionando as opções “New > Other > JavaScript > JavaScript Source File” e chame-o de “js\_java\_exemplo.js”, para seguir o padrão de nomenclaturas adotado nas listagens destes exemplos. Insira no mesmo o código presente na **Listagem 5**.

#### Listagem 5. Código do arquivo de teste de JavaScript

```
function ola() {
    var ola = 'DEVMEDIA'.toLowerCase();
    itera();
    print('Ola Mundo' + ola + '!');
}
function itera() {
    var cont = 1;
    for (var i = 0, max = 5; i < max; i++) {
        cont++;
    }
    print('Valor da var cont:' + cont);
}
ola();
```

Observe agora a comunicação entre o código contigo na **Listagem 4**, referente às chamadas do arquivo de mesmo nome criado agora na **Listagem 5**. Isso mostra a conformidade entre chamadas físicas ao arquivo tanto no lado servidor/Java quanto em um possível universo web, com a inclusão de chamadas pelo browser no lado cliente.

Basicamente temos duas funções no arquivo como um todo (totalmente código JavaScript): a primeira, a função ola(), se encarrega de criar uma nova variável com um valor literal todo em maiúsculo e logo em seguida chama a função “toLowerCase()” que será responsável por traduzir o texto para sua versão em minúsculo. Em seguida, chamamos a segunda função, itera(), que criará uma variável contadora e irá iterar seu valor até 6, imprimindo-o no final. Finalmente, a execução volta e imprime a mensagem de boas vindas formatada.

Interessante notar que toda essa execução findará no resultado abaixo, somente porque efetuamos a chamada ao método “ola()” no final do mesmo arquivo js:

Saída do Console:

Valor da var cont: 6

Ola Mundo devmedia!

O mesmo resultado poderá ser verificado quando da execução do segundo método da classe Teste, executarViaCodigoInline(), uma vez que o mesmo contém a mesma linha de execução, porém portando todo o código inline. Quando a execução exigida for a presente no primeiro método, neste caso é obrigatório o uso do método JavaScript “load()” para que possa assim carregar o arquivo js apropriadamente.

Adicionalmente, veja que os métodos estão cercados pela cláusula try/catch em vista da possibilidade de haver alguma exceção do tipo ScriptException. Em nível de teste rápido, mude o nome do arquivo JavaScript no próprio ou apenas na chamada ao load. Você deverá receber uma exceção do tipo:

```
javax.script.ScriptException: TypeError: Cannot load script from src/java_js_exemplo2.js
in <eval> at line number 1
```

#### Chamando Java a partir do JavaScript

Efetuar chamadas do Java a partir de arquivos JavaScript é tão fácil quanto o inverso. O único conceito a se assimilar (e se acostumar, pois realmente é bem incomum) é de efetuar a chamada explícita dos códigos Java de dentro do arquivo js. Isso porque as classes das bibliotecas Java 8 estão construídas dentro do Nashorn. Crie um novo arquivo js também na pasta src e nomeie-o como “java\_js\_exemplo.js”. Considere a inclusão do código apresentado na **Listagem 6** no mesmo.

#### Listagem 6. Código de teste para executar Java a partir de JavaScript

```
// Recupera o tempo corrente em milissegundos
print(java.lang.System.currentTimeMillis());
// Imprime valores absolutos do diretório JS
var file = new java.io.File("js_java_exemplo.js");
print(file.getAbsolutePath());
print(file.getAbsolutePath());
```

Nesse código podemos observar duas implementações básicas: uma chamada à classe java.lang.System (Sim, precisamos importar sempre as classes Java que serão usadas) para impressão do valor atual do tempo em milissegundos, e uma chamada ao arquivo “js\_java\_exemplo.js” que se encontra no mesmo diretório relativo para impressão do seu caminho absoluto. Note que existem duas chamadas diferentes aos valores do “absolutePath”, demonstrando que não é necessário fazer chamadas explícitas aos métodos getters e setters dos objetos em JavaScript, isto é, o JavaScript converte automaticamente o atributo aos métodos get's e set's quando da conversão para bytecode. Perceba também que o Nashorn não importa o pacote Java por padrão, porque as referências a String ou Object podem gerar conflito com os tipos correspondentes no JavaScript. Assim, uma String em Java é sempre representada por java.lang.String, e não somente String.

Para testar o exemplo anterior, basta mudar a chamada no método “executarViaArquivoJS()” para:

```
engine.eval("load(\"src/js_java_exemplo.js\");");
```

Dessa forma, ao executar, teremos um resultado semelhante ao descrito a seguir sendo impresso no Console:

1402875392174 – O valor varia de acordo com a hora da execução

C:\Users\Julio\nashorn-devmedia\js\_java\_exemplo.js

C:\Users\Julio\nashorn-devmedia\js\_java\_exemplo.js

O ponto chave de toda a execução é levar em consideração que o meio comum entre as duas implementações está na função “print()”, reconhecida por ambas linguagens. Assim, é possível averiguar os resultados em ambos os lados.

As implementações podem ir além, exigindo mais poder de processamento, como a inclusão de conversores e formatadores, bem comuns à linguagem Java quando se considera as regras de negócio dos sistemas, bem como a necessidade das mesmas. Efetue a mudança do conteúdo do arquivo js java\_js\_exemplo.js para o apresentado na [Listagem 7](#).

## Listagem 7. Código de teste para formatações usando JavaScript

```
// Recupera objeto e exibe formatação de data padrão brasileiro
print(new java.text.SimpleDateFormat("dd/MM/yyyy").format(new java.util.Date()));

// Recupera objeto e exibe formatação de número decimal
print(new java.text.DecimalFormat("###.##").format(345.345));
```

Ao executar o código em questão, você terá um resultado como, de acordo com o dia que estiver a rodar o mesmo:

17/06/2014

345,35

Dessa forma, é possível executar ações mais complexas e que exigam explicitamente mais processamento direto da própria JVM.

Como mencionado anteriormente, um dos recursos mais poderosos do Nashorn é a possibilidade de chamar classes Java de dentro do JavaScript. Você pode não apenas acessar classes e criar instâncias, mas também pode criar subclasses delas, chamar seus membros estáticos, e fazer praticamente qualquer coisa que você poderia fazer a partir de Java.

Para exemplificar, vejamos o velho recurso das Threads no Java. O JavaScript não tem os recursos de linguagem para concorrência e todos os runtimes comuns são single-thread ou pelo menos sem qualquer estado compartilhado. É interessante ver que, no ambiente Nashorn, o JavaScript poderia, de fato executar simultaneamente e com o estado compartilhado, assim como no Java. Para conferir tal comportamento, crie um novo arquivo JavaScript e nomeie-o de “java\_threads.js”, e adicione em seguida o código contigo na [Listagem 8](#) ao mesmo.

Observe a presença de duas funções distintas no exemplo anterior. Ambas retratam usos diferentes do recurso de Threads no Java: a primeira explicita o uso do método sleep() para pausar uma execução na thread corrente pelo tempo passado por parâmetro

em milissegundos, enquanto a segunda mostra um exemplo fiel de como criar um thread em Java, com a implementação sobreescrita do seu método run() bem como a respectiva chamada ao método start() da mesma, garantindo, assim, a execução em paralelo dos dois códigos. Repare também que a forma canônica para acessar uma classe do Nashorn é usando Java.type e você pode estender uma classe usando Java.extend.

## Listagem 8. Código de teste para verificar o uso de Threads no JavaScript

```
function loop() {
    for (i = 0; i < 5; i++) {
        print("Quantidade de tentativas: " + i);
        java.lang.Thread.sleep(2000);
    }
    iniciarThread();
}
loop();

function iniciarThread() {
    // Esse código acessa a classe Java Thread diretamente
    var Thread = Java.type("java.lang.Thread");

    // Subclasse que iremos usar para chamar o método run
    var MyThread = Java.extend(Thread, {
        run : function() {
            print("Executou em uma Thread separada!");
        }
    });
    var th = new MyThread();
    th.start();
    th.join();
}
```

O resultado da execução desse código será a impressão, pausada em dois segundos cada, da mensagem informando a quantidade de tentativas atualizada pela variável contadora, assim como a mensagem do segundo método de execução paralela.

Isso mostra que o poder de comunicação entre as linguagens é perpassado e vai de encontro justamente ao que ambas tem de mais comum nesse novo ambiente de execução: a geração final de bytecode Java.

## Passando dados para e a partir do Java

Como indicado nos exemplos anteriores, você pode chamar código JavaScript diretamente do seu código Java; para tal, basta obter a engine e chamar o seu método “eval”. Além disso, você também pode passar dados explicitamente como strings ou passar bindings de Java que podem ser acessados como variáveis globais de dentro da engine JavaScript.

Veja na [Listagem 9](#) exemplos comentados de como isso poderia ser feito facilmente. Para executar os exemplos basta realizar o mesmo procedimento dos anteriores.

## Listas e Collections

Certamente, um dos recursos mais favoritos e usados da linguagem Java em contraste às suas linguagens “fundadoras” ou anteriores, é o uso da API de Collections e suas hierarquias. Nada

mais justo que entender também como funciona a comunicação das duas linguagens em relação a essa famosa estrutura de dados, no que concerne aos conceitos de programação.

Por todos os aspectos, com o lançamento do JDK 8, Java tem, pelo menos até certo ponto, se tornado uma linguagem funcional. Agora você pode usar funções de ordem superior em coleções, por exemplo, para iterar sobre seus elementos. A função de ordem superior é uma função que recebe outra função como um parâmetro e faz algo significativo com ele. Vamos analisar o código contigo na **Listagem 10** e discorrer sobre o assunto.

#### **Listagem 9.** Código de teste para passagem de valores entre as duas linguagens

```
// Método de execução básica a partir do Java
private void testeValoresAPartirDoJava() {
    try {
        ScriptEngineManager scriptEngineManager = new ScriptEngineManager();
        ScriptEngine engine = scriptEngineManager.getEngineByName("nashorn");
        String nome = "Julio";
        // Os valores em Java podem ser passados em associação comum de strings
        engine.eval("print('Olá, " + nome + "! Tudo bem?')");
    } catch (ScriptException e) {
        e.printStackTrace();
    }
}

// Método de execução básica a partir do JavaScript
idade = 30;
// O valor é recuperado e passado para um objeto SimpleBindings
SimpleBindings bindingSimples = new SimpleBindings();
simpleBindings.put("globalValue", idade);
// No momento de traduzir a expressão, o binding é feito
nashorn.eval("print (globalValue)", bindingSimples);
```

#### **Listagem 10.** Exemplo de uso das funções de ordem superior

```
@SuppressWarnings({"unchecked", "rawtypes"})
private void testeFuncaoOrdemSuperior() {
    List<? extends Object> listaGenerica = Arrays.asList("ABC", 4, new Object(), 2.45);
    listaGenerica.forEach(new Consumer() {
        @Override
        public void accept(Object object) {
            System.out.println(object);
        }
    });
}
```

Neste exemplo, em vez de iterar sobre os elementos usando um loop “externo” como seria tradicionalmente feito, agora passamos uma função “Consumer” para a operação de “forEach”, uma operação de ordem superior “internal-loop”, que executa o método “accept” do Consumer passando em cada elemento da lista, um por um.

Como já mencionado, a abordagem da linguagem funcional para tal função de ordem superior aceitaria sim um parâmetro de função, em vez de um objeto. Passar referências às funções por si só não é algo tradicionalmente Java, mas o JDK 8 agora tem recursos sintáticos mais elegantes para sintetizar o uso de expressões lambda. Por exemplo:

```
List<? extends Object> listaGenerica = Arrays.asList("ABC", 4, new Object(), 2.45);
listaGenerica.forEach(elemento -> System.out.println(elemento));
```

Neste caso, o parâmetro de “forEach” tem a forma de uma função de referência. Isto é possível porque o Consumer agora é uma interface funcional, (às vezes chamado de um tipo Single Abstract Model, ou “SAM”).

Então por que estamos falando de lambdas em um artigo sobre Nashorn? Porque em JavaScript você também pode escrever código como este e Nashorn é especialmente bem preparado para fazer a ponte entre Java e JavaScript, neste caso. Em particular, ele permite que você mesmo passe funções JavaScript simples como implementações das interfaces funcionais (tipos SAM).

Vamos dar uma olhada no mesmo código escrito na **Listagem 10**, porém agora em JavaScript, que faz a mesma coisa que o nosso código Java anterior. Note que não há nenhum tipo de lista embutida no JavaScript, apenas arrays; mas esses arrays são dimensionados de forma dinâmica e tem métodos comparáveis aos de uma lista Java. Assim, neste exemplo, estamos chamando o método “forEach” de um array JavaScript (**Listagem 11**).

#### **Listagem 11.** Tradução do uso das funções de ordem superior no JavaScript

```
function testeFuncaoOrdemSuperior() {
    var listaGenerica = ['ABC', 4, new java.lang.Object(), 2.45];
    listaGenerica.forEach(function(elemento) {
        print(elemento);
    });
    // Ou...
    var lista = java.util.Arrays.asList(listaGenerica);
    lista.forEach(function(elemento) { print(elemento) });
}
testeFuncaoOrdemSuperior();
```

O Nashorn permite-nos fornecer referências das funções JavaScript simples onde se espera uma interface funcional (tipo SAM). Este é, portanto, não somente possível a partir do Java, mas também a partir do JavaScript.

A próxima versão do ECMAScript, que deverá se tornar definitiva este ano, irá incluir uma breve sintaxe para as funções que lhes permitam ser escrita quase como lambdas Java, exceto que ele usará uma seta dupla =>.

## Avatar.js

Vimos que com o Nashorn temos uma engine JavaScript Premium embutida no Java. Vimos também que a partir do Nashorn podemos acessar qualquer classe Java. O Avatar.js vai um passo além e traz “o modelo de programação Node, APIs e módulo de ecossistema para a plataforma Java.” Para entender o que isso significa e por que é emocionante, primeiro temos que entender o que é o Node. O Node basicamente extrai a engine V8 JavaScript do Chrome para que ela funcione a partir da linha de comando sem a necessidade de um navegador. Isso faz do JavaScript executável não só no navegador, mas também no lado do servidor. Para executar JavaScript em um servidor de forma significativa pelo menos você vai precisar acessar o sistema de arquivos e a rede. Para conseguir isso, o Node embute uma biblioteca chamada libuv que faz isso de forma assíncrona.

Praticamente, isto significa que as chamadas para o sistema operacional nunca são bloqueadas, mesmo se elas demorarem um pouco para voltar. Em vez de bloquear, você fornece uma função de callback que será acionada assim que a chamada é feita, entregando os resultados, se houver algum.

Vejamos, portanto, um exemplo extraído diretamente do site do NodeJS ([Listagem 12](#)).

## Nota

O foco deste artigo não será em nenhum momento explanar acerca do NodeJS ou quaisquer tecnologias semelhantes. Pressupõe-se que o leitor tenha tais conhecimentos ou busque os mesmos. Nessa edição temos dois artigos que tratam desse assunto.

## Listagem 12. Exemplo de uso do NodeJS

```
// carrega o módulo "http" para lidar com as requisições http
var http = require('http');

// Quando há uma requisição retorna a mensagem 'Hello, World\n'
function handleRequest(req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello, World\n');
}

/*
Estamos ouvindo no localhost, na porta 1337 e
chamamos o handleRequest como call back
você verá uma natureza non-blocking/asynchronous nessa implementação
*/
http.createServer(handleRequest).listen(1337, '127.0.0.1');

// Escreve no console para garantir que estamos no caminho certo
console.log('Get your hello at http://127.0.0.1:1337');
```

Conforme mencionamos anteriormente, você necessitará instalar o NodeJS para verificar a execução desse exemplo. Aqui nos ateremos a demonstrar como funciona a comunicação entre ambas as tecnologias, ficando a cargo do autor se aprofundar na linguagem caso queira.

O objetivo do Avatar.js é fornecer o mesmo conjunto de APIs core que o Node apresenta, através da construção de classes Java usando a famosa “libuv” e fazendo-as, então, acessíveis via JavaScript. Apesar desse tipo de implementação parecer um pouco estranha e rústica, ela costuma funcionar muito bem. O Avatar.js suporta um grande número de módulos NodeJS além do suporte nativo ao “express”: uma stream principal do framework web para o Node, o que indica, sobretudo, que esse tipo de recurso pode trabalhar em conjunto com um grande número de projetos já existentes.

Infelizmente, no momento de escrita deste artigo não existe nenhuma distribuição binária para o Avatar.js para o contexto de implementação que sugerimos. Se acessar a página do projeto poderá ver um arquivo readme que explica como construir tudo a partir do código fonte. Uma vez que você tenha instalado os binários e os colocado dentro da pasta lib, você poderá então chamar o Avatar.js usando algo como:

```
java -Djava.library.path=lib -jar lib/avatar.js.jar helloWorld.js
```

E mais uma vez podemos nos perguntar... Mas onde isso tudo pode ser útil? A Oracle vê uma série de casos de uso para essa biblioteca, bem como várias situações de usabilidade para a mesma dentro dos diferentes contextos de implementação. Dentre as opções oficiais, duas se destacam:

- Você tem uma aplicação NodeJS e quer usar algumas bibliotecas externas para complementar a API;
- Você quer mudar para o JavaScript e para a API do Node, mas você tem código legado em Java que não funcionaria totalmente fora de um ambiente com JVM.

Em ambas situações funcionam bem com o Avatar.js, além de poder fazer qualquer chamada a classes Java a partir do código JavaScript, que é suportado naturalmente pelo Nashorn, como vimos até então.

Vamos supor um cenário em que, como sabemos, o JavaScript tenha apenas um simples tipo para expressar números: o tipo “number”. Isso seria equivalente ao “double” do Java, no quesito precisão e limitações. Números em JavaScript, assim como os doubles em Java, não são aptos para expressar intervalos arbitrários e precisão, quando por exemplo lidamos com cálculos sobre valores monetários.

Em Java, você pode usar a classe BigDecimal, que contém algumas dúzias de métodos que te ajudarão a lidar com toda essa situação facilmente, ou até mesmo usar bibliotecas terceiras para resolver o problema. Mas já o JavaScript não tem nada equivalente em seu core, então você pode simplesmente pegar emprestado o uso da classe BigDecimal a partir do seu código JavaScript e ter, dessa forma, o mesmo comportamento que teria no Java.

Vejamos o exemplo ilustrado na [Listagem 13](#), onde temos um serviço web que calcula o percentual sobre uma determinada quantia monetária.

Em JavaScript não existem tipos declarados, mas se observar bem os dois primeiros métodos da listagem, perceberá a extrema semelhança entre as implementações em JavaScript e Java, respectivamente.

Para efetuar o teste integrado basta adicionar a mesma à [Listagem 12](#) anterior, modificando as partes devidas para tal, tal como ilustrado no método handleRequest da [Listagem 13](#). Dentro de um ambiente montado e funcional, o teste poderá ser efetuado acessando uma URL como:

```
http://localhost:1337/calcular?quantia=1943534500534534000008654613&percentual=6.45
```

O resultado seria a impressão do valor calculado no próprio browser. Analisando o ponto de vista de usabilidade para essa tecnologia, teríamos algumas outras onde isso se aplicaria tal como quando você decide migrar sua aplicação JEE existente para JavaScript e Node. Nesse caso, você pode facilmente acessar todos os seus serviços existentes a partir do JavaScript. Outro caso de uso seria ter uma nova funcionalidade no servidor escrita em JavaScript e Node que ainda poderia ser beneficiada a partir dos serviços JEE existentes.

**Listagem 13.** Exemplo de uso das classes em JavaScript para cálculos monetários

```
var BigDecimal = Java.type('java.math.BigDecimal');

function calcularPercentual(quantia, percentual) {
    var resultado = new BigDecimal(quantia).multiply(
        new BigDecimal(percentual)).divide(
        new BigDecimal("100"), 2, BigDecimal.ROUND_HALF_EVEN);
    return resultado.toPlainString();
}

public static String calcular(String quantia, String percentual) {
    BigDecimal resultado = new BigDecimal(quantia).multiply(
        new BigDecimal(percentual)).divide(
        new BigDecimal("100"), 2, BigDecimal.ROUND_HALF_EVEN);
    return resultado.toPlainString();
}

// carrega o módulo utility'url' para url
var url = require('url');

function handleRequest(req, res) {
    // 'calcular' é o path do nosso web service
    if (url.parse(req.url).pathname === '/calcular') {
        var query = url.parse(req.url, true).query;
        // quantia and percentual are passed in as query parameters
        var resultado = calcularPercentual(query.quantia,
            query.percentual);
        res.writeHead(200, {'Content-Type': 'text/plain'});
        res.end(resultado + '\n');
    }
}
```

Indo na mesma direção podemos encontrar ainda o Projeto Avatar que é baseado no Avatar.js. A ideia básica é escrever sua aplicação em JavaScript e acessar os serviços JEE. O Projeto Avatar vem com uma distribuição binária combinada para o Avatar.js, mas requer o Glassfish para instalação e desenvolvimento, além de servidor base.

## Nashorn e JavaFX

O JavaFX é um conjunto de gráficos e pacotes de mídia que possibilita aos desenvolvedores criar, estilizar, testar, debugar e iniciar aplicações ricas no cliente e que operam consistentemente sobre diversas plataformas.

Com o Nashorn, você pode interpretar um script JavaFX usando o comando jjs com a opção -fx na linha de comando. Vejamos o exemplo desenvolvido na **Listagem 14**.

Para executar esse exemplo, crie um novo arquivo chamado "javafx\_testes.js" dentro do diretório onde se encontra o .bat do jjs, isto é, na pasta bin do seu JDK 8, adicione o código da **Listagem 14** ao mesmo e execute o comando a seguir, tal como ilustrado na **Figura 5**.

```
jjs -fx -scripting javafx_testes.js
```

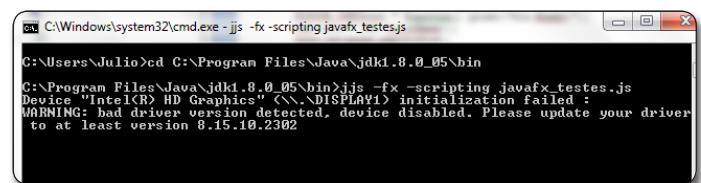
O resultado da execução será semelhante ao exibido na **Figura 6**.

Note também que ao efetuar o clique no botão disponibilizado, a mensagem "Ola Mundo!" será exibida no Console do seu prompt.

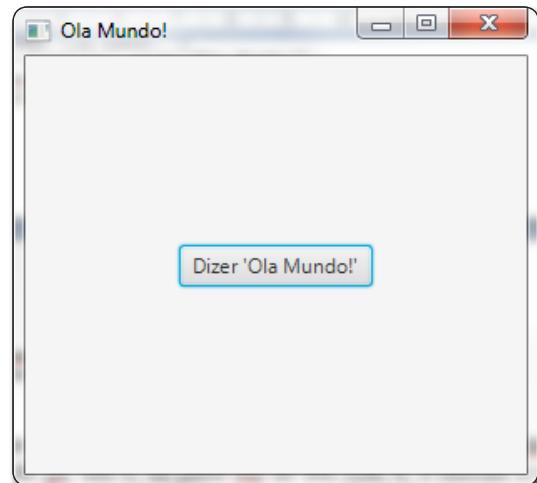
**Listagem 14.** Exemplo de uso do Nashorn para chamar código JavaFX

```
var Button = javafx.scene.control.Button;
var StackPane = javafx.scene.layout.StackPane;
var Scene = javafx.scene.Scene;

function start(estagiolnicial) {
    estagiolnicial.title = "Ola Mundo!";
    var button = new Button();
    button.text = "Dizer 'Ola Mundo!'";
    button.setOnAction = function() print("Ola Mundo!");
    var root = new StackPane();
    root.children.add(button);
    estagiolnicial.scene = new Scene(root, 300, 250);
    estagiolnicial.show();
}
```



**Figura 5.** Execução do script JavaFX no prompt de comando



**Figura 6.** Resultado da execução do script de JavaFX com Nashorn

## Debugando Nashorn

Uma das desvantagens do Nashorn é que o mesmo não inclui nenhum tipo de ferramenta de auxílio a debug. Felizmente, tanto o NetBeans 8 quanto o IntelliJ IDEA a partir da sua versão 13.3 suportam esse tipo de funcionalidade via JavaScript. Os recursos são facilmente encontrados e de boa usabilidade, concentrando-se num item de menu que abre uma pop-up nos arquivos JavaScript permitindo que você debugue seu código.

No IntelliJ você pode configurar breakpoints nos arquivos Java e JavaScript usando atalhos também. Quando você configura breakpoints automaticamente assume todo o poder de debugging usual.

O Nashorn é uma tecnologia que foi desenhada para ser um substituto mais rápido e melhor para a antiga engine Rhino, e até então tem mostrado sucesso nessa caminhada.

## Nashorn: Desenvolvendo JavaScript na JVM

A tecnologia ainda não se encontra no seu nível ideal, muitas correções precisam ser feitas e bugs precisam ser analisados e reparados. Porém, para o que há até então, com certeza a tecnologia supera as expectativas e vai além, permitindo flexibilidade inclusive na migração de códigos dessa natureza, envolvendo as tão famosas e importantes linguagens Java e JavaScript.

O futuro ainda reserva muito para as tecnologias em questão, e certamente o Nashorn irá estar presente, melhorando cada vez mais.

### Autor



Júlio Sampaio

Analista de sistema e entusiasta da área de Tecnologia da Informação. Atualmente é consultor na empresa Visagio, trabalhando em projetos de desenvolvimento de sistemas estratégicos, é também instrutor JAVA. Possui conhecimentos e experiência em áreas como Engenharia de Software e Gerenciamento de Projetos, tem também interesse por tecnologias relacionadas ao front-end web.



### Links:

**Página oficial do Projeto**

[www.openjdk.java.net/projects/nashorn/](http://www.openjdk.java.net/projects/nashorn/)

**Especificação ECMAScript 5.1**

[www.ecma-international.org/ecma-262/5.1/](http://www.ecma-international.org/ecma-262/5.1/)

**Site do Projeto Avatar.**

[www.avatar.java.net/](http://www.avatar.java.net/)

# CURSOS ONLINE

A Revista Clube Delphi oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.



### CONHEÇA ALGUNS DOS CURSOS:

- **Curso de Multicamadas com Delphi e DataSnap**
- **Delphi para Iniciantes**
- **Criando componente Boleto em Delphi**
- **Loja Virtual em Delphi Prism**

Para mais informações :

[www.devmedia.com.br/cursos/delphi](http://www.devmedia.com.br/cursos/delphi)

(21) 3382-5038



**DEV**MEDIA

# JavaScript Blog: Criando serviço de microblog com NodeJS

Conheça os recursos da plataforma mais famosa em escalabilidade front-end

Para aqueles que não conhecem ainda a plataforma NodeJS, à primeira vista pode parecer que se trata de mais uma biblioteca JavaScript para uma série de soluções e facilidades no front-end de websites, tal como o jQuery, o Dojo, dentre outros. Mas na verdade o NodeJS se trata de uma plataforma server-side que promete reduzir a carga de processamento na máquina do servidor através de uma arquitetura que atende a todas as requisições feitas ao servidor de forma que não ocorram bloqueios de I/O e livre de deadlocks. Além disso o NodeJS permite um maior controle do servidor por parte do desenvolvedor por este se tratar de uma plataforma em linguagem baixa.

Em comparação com os servidores comuns, em cada conexão, é alocado um espaço de memória. As requisições de cada usuário são enfileiradas e processadas na devida ordem, o que gera certo atraso na resposta da requisição, porque as requisições não são processadas em paralelo, ou seja, o servidor por sua vez é sobrecarregado e a eficiência do serviço ou sistema é comprometida, isto no caso de um aumento de tráfego e normalmente este problema é contornado adicionando mais servidores ou adicionando mais recursos de hardware para aumento da memória e maior capacidade de processamento.

Para solucionar este tipo de problema, foi criado o NodeJS, uma plataforma de software criada em 2009 por Ryan Dahl através do uso da engine V8 JavaScript. A principal vantagem do NodeJS é exatamente o processamento das requisições não se bloquearem e não entrarem em uma fila de processamento. Isso acontece porque cada nova conexão ao servidor irá criar um novo processo isento de qualquer alocação de espaço na memória.

## Fique por dentro

Nesta publicação serão descritos os principais conceitos da plataforma NodeJS, sua aplicabilidade e um breve tutorial para elaboração de um aplicativo de micro blog, demonstrando todo o potencial da biblioteca Express e de outros módulos que são utilizados rotineiramente no desenvolvimento de sistemas de protocolo HTTP. Este tema é útil para a elaboração de aplicações ou módulos de sistemas que estejam sujeitos à sobrecarga de acessos ou de processamento, visando evitar que sejam necessárias melhorias de hardware para suportar a carga da aplicação. Além de atender a necessidade da implementação de um sistema em tempo real, através de uma conexão bidirecional (ou keep-alive).

Logo, o sistema tem um tempo de resposta muito mais ágil e economia no uso do hardware e melhor aproveitamento do processamento do servidor. Devido à arquitetura orientada a eventos do JavaScript cada processo do server-side será disparado também através de eventos, mas desta vez não serão eventos onClick ou load, mas sim através de eventos de conexão, de requisição, dentre uma série de outras possibilidades.

Pelo fato do NodeJS suportar os mais diversos protocolos como SSH, FTP, SMTP, DNS e dentre outros, as possibilidades de desenvolvimento do Node se estendem desde aplicações web simples no protocolo HTTP até servidores para jogos multiplayer. O NodeJS a partir da sua versão 0.6.3 passou a incluir de forma nativa o seu instalador de dependências, o NPM (Node Package Manager), um sistema auxiliar que ao ler o arquivo package.json, que contém as referências do projeto devidamente listadas, realiza o download de cada uma das dependências.

## Preparando o ambiente

Para começarmos nosso desenvolvimento vamos inicialmente instalar o NodeJS e logo após o NPM e o banco de dados que utilizaremos neste artigo, no caso, o MongoDB. Para este, iremos utilizar a versão 0.10.21, a mais recente até a data da redação deste artigo.

Para instalar o NodeJS nos sistemas operacionais Mac OS e Windows basta acessar o site oficial do NodeJS (na seção **Links**), baixar o instalador para o seu sistema operacional e executá-lo.

Mas para o caso do Linux, há outra forma para instalá-lo, no caso da distribuição Ubuntu, versão 13, ele vem por padrão no *apt-get* com a versão 0.6 (um pouco desatualizada para este artigo) para isto então se deve executar os comandos apresentados na **Listagem 1**.

**Listagem 1.** Sequência de comandos para a instalação do NodeJS no Linux.

```
sudo apt-get update
sudo apt-get install -y python-software-properties python g++ make
sudo add-apt-repository -y ppa:chris-lea/node.js
sudo apt-get update
sudo apt-get install nodejs
```

Caso a sua instalação não tenha sido bem sucedida ou você prefira instalar o NodeJS pelo gerenciador de pacotes do Linux nas mais diferentes distribuições, todos os sites com as informações de download podem ser encontradas na seção **Links**.

Antes de configurar o ambiente, verifique se o NodeJS foi instalado com sucesso, para isto, acesse o terminal (ou prompt) e execute o comando *node -v* que em resposta ao programa irá retornar a versão instalada do NodeJS, como demonstrado na **Figura 1**.

```
milleo@webmagazine:~$ nodejs -v
v0.10.21
```

**Figura 1.** Verificando versão instalada do NodeJS

Se tudo deu certo, vamos fazer um teste no REPL (*Read Eval Print Loop*), ou seja, um *interactive shell* similar ao *irb* do Ruby, um sistema interativo que executa as linhas de código inseridas no terminal. Primeiro, escreva *node* no terminal, tecle enter e insira o seguinte comando:

```
console.log("Olá NodeJS");
```

E tecle enter, o programa irá responder com a mensagem “Olá NodeJS” e para sair do REPL tecle *ctrl + c*.

Após a instalação, deve-se configurar a variável de ambiente no seu sistema operacional (isto vale para Windows, MAC OS e Linux), no caso do MAC OS e o Linux basta configurar o arquivo *.bash\_profile* ou *.bashrc* (acessível de dentro do diretório *home* e caso o arquivo ainda não exista, você deverá criá-lo) e adicionar a seguinte linha ao final do arquivo:

```
NODE_ENV="development"
```

Já no Windows, a configuração da variável de ambiente se dá da seguinte forma:

- Clique em iniciar;
- Com o botão direito selecione a opção “Computador”;
- Na janela de configurações, dentro da seção de “Nome do computador (...)” clique em “Alterar Configurações”;
- Selecione a aba “Avançado” e clique no botão “Variáveis de ambiente”;
- Abaixo da box “Variáveis do Sistema” clique em “Novo” e preencha o campo Nome da variável com “NODE\_ENV” e o campo “Valor da variável” com “development”.

Após a execução deste passo a passo o Windows já está preparado para o desenvolvimento em NodeJS!

## Criando o primeiro servidor

Antes de começarmos com o projeto principal vamos antes criar uma pequena aplicação para entendermos como é feita a instanciação de um servidor HTTP com NodeJS. Antes crie um diretório, onde preferir, para armazenarmos os nossos projetos e crie um diretório para este projeto, chamando-o de **PrimeiraAplicacao** e em seguida crie um arquivo chamado *app.js* dentro dele com o conteúdo apresentado na **Listagem 2**.

**Listagem 2.** Primeiro script de um servidor HTTP simples.

```
var http = require('http');
var server = http.createServer(function (req, res){
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("<h1>Seja bem-vindo!</h1>");
  res.end();
});
server.listen(1337);
```

Salve o arquivo, abra o terminal (ou prompt) e acesse através do terminal o diretório onde o arquivo foi salvo. Execute o arquivo e inicialize o servidor HTTP, basta digitar o seguinte comando no terminal:

```
node app.js
```

Logo após abra o seu navegador e vamos acessar o seguinte endereço: *http://localhost:1337/* para verificar se o resultado será semelhante ao apresentado na **Figura 2**.



**Figura 2.** Resultado da implementação de um servidor HTTP simples

Ao analisar o código da implementação deste pequeno servidor, percebe-se na primeira linha a importação do módulo HTTP para a criação de um servidor que atenda a este protocolo. Logo após declara-se uma variável server e a esta variável é atribuído o resultado do método *createServer* do objeto HTTP, este método é alimentado pelas variáveis de requisição e resposta (req e res, respectivamente) e no conteúdo do call-back inicia-se a escrita da resposta. No início, é escrito o cabeçalho da resposta atribuindo um código de resposta 200, representando sucesso na requisição, e o tipo de resposta, neste caso, texto HTML. Em seguida escrevemos o conteúdo que aparecerá na tela, a mensagem “Seja bem-vindo” e logo após é definido o fim da transmissão de dados através do método end. Finalmente, é atrelada a função *listen*, que fará o acionamento da função do servidor caso alguma requisição chegue à porta que foi inserida no parâmetro da função *listen*.

Porém neste primeiro exemplo, não foi explorado o objeto de requisição recebido pelo servidor, no próximo exemplo este parâmetro será mais explorado, para isto, interrompa a execução do primeiro script no terminal pressionando *ctrl + c* (ou *command + c*) e altere o arquivo *app.js* tal como na **Listagem 3**.

**Listagem 3.** Adicionando suporte a URLs ao servidor HTTP.

```
var http = require('http');
var server = http.createServer(function(req, res){
  res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
  if(req.url == "/"){
    res.write("<h1>Ola NodeJS</h1>");
  }else if(req.url == "/outra/"){
    res.write("<h1>Outra página</h1>");
  }else{
    res.writeHead(404, {'Content-Type': 'text/html; charset=utf-8'});
    res.write("<h1>Página não encontrada</h1>");
  }
  res.end();
});
server.listen(1337);
```

Ao finalizar salve o arquivo, retorne ao terminal e execute novamente o código no terminal:

```
node app.js
```

Ao executar, vá até o navegador e acesse o endereço *http://localhost:1337/* para receber a mensagem de boas-vindas como visto no exemplo anterior, logo após acesse o endereço *http://localhost:1337/outra/* para receber a mensagem “Outra página” e por último insira uma URL inexistente tal como *http://localhost:1337/erro/* para ter o retorno da página de erro previamente escrita, o resultado deve ser como apresentado na **Figura 3**.

Em análise, nota-se que agora o código conta com três estruturas condicionais realizando uma verificação no atributo URL do objeto req, este atributo traz apenas o path da requisição (Por exemplo: *http://www.site.com.br/caminho/do/path* retorna o valor */caminho/do/path/*), dentro do objeto de requisição encontramos outros atributos para exibir mais valores específicos da URL.



**Figura 3.** Página de erro apresentado pela aplicação

## Criando um Microblog

Para auxiliar no aprendizado desta plataforma, será criado um passo a passo de uma aplicação web simples de microblog similar ao Twitter, com algumas funções limitadas. E para isto será utilizado o framework Express logo que a atividade de desenvolvimento de uma aplicação HTTP deste porte pode se tornar uma atividade muito desgastante. Também será utilizado o banco de dados no-SQL MongoDB devido a sua fácil integração com o NodeJS.

Para iniciar abra o terminal (ou prompt) e execute o código:

```
npm install -g express
```

No Linux é necessário executar este código no modo sudo (insira a notação sudo antes do comando). Logo após executar o código e a instalação ser finalizada, é necessário reiniciar o terminal para que as alterações surtam efeito no ambiente de desenvolvimento. Ao reiniciar o terminal é necessário acessar através do mesmo diretório onde se deseja implementar o projeto (escolha um a seu gosto) e em seguida execute o código no terminal:

```
express nwwitter
```

Quando o código é executado, a saída do programa será similar a **Figura 4** em seu terminal.

Após executar o comando, é criada uma estrutura de diretórios do projeto Express, esta estruturação é padrão do próprio framework. Para concluir a criação acesse através do terminal o diretório que foi criado. Por fim, execute o seguinte comando:

```
npm install
```

Quando o projeto foi gerado, o *package.json* já foi criado com duas dependências: o Express propriamente dito e o Jade, um template engine que utiliza notações mais diretas e identação na geração de tags HTML. Isto fará com que todas as dependências sejam instaladas no projeto. Acessando o diretório do projeto pode-se perceber a seguinte estrutura de diretório e arquivos:

- **app.js** – Arquivo principal do projeto, onde o servidor será instanciado e inicializado;
- **package.json** – Todas as informações do projeto e suas dependências são contempladas neste arquivo, tal como, o nome do projeto, versão, autor, licença, e dentre várias outras informações. ([Links](#));

- **/node\_modules/** – Diretório onde todas as dependências ficam alocadas;
- **/public/** – Local padrão para armazenar imagens, scripts, folhas de estilo e etc.;
- **/routes/** - Todos os scripts de rotas são guardados neste diretório;
- **/views/** - Onde devemos armazenar todos os arquivos da camada de view do projeto, scripts que geram o layout.

```
milleo@webmagazine: ~/NodeJS
File Edit View Search Terminal Help
milleo@webmagazine:~/NodeJS$ express nwwitter

  create : nwwitter
  create : nwwitter/package.json
  create : nwwitter/app.js
  create : nwwitter/public
  create : nwwitter/public/javascripts
  create : nwwitter/public/images
  create : nwwitter/public/stylesheets
  create : nwwitter/public/stylesheets/style.css
  create : nwwitter/routes
  create : nwwitter/routes/index.js
  create : nwwitter/routes/user.js
  create : nwwitter/views
  create : nwwitter/views/layout.jade
  create : nwwitter/views/index.jade

install dependencies:
$ cd nwwitter && npm install

run the app:
$ node app
```

Figura 4. Saída gerada pelo Express ao criar um projeto

Porém, para o projeto também é necessário criar dois diretórios um para todos os controles e outro diretório para armazenar todos os modelos, portanto, acesse o diretório do projeto e crie um diretório controllers e outro models. Antes de começar a construir o projeto deve-se antes de tudo preencher o arquivo package.json com os dados corretos para o projeto, altere-o de forma que se assemelhe a **Listagem 4**.

Listagem 4. Arquivo package.json.

```
{
  "name": "nwwitter",
  "description": "Minha primeira aplicação NodeJS",
  "autor": "Seu Nome <seu@email.com>",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "3.4.4",
    "jade": "*"
  }
}
```

Mas para este tutorial o Jade será substituído pelo EJS, por ser um template engine que utiliza HTML e tags especiais para o código JS, bastante similar ao PHP. E para isto é necessário desinstalar as dependências do Jade, abra o arquivo package.json em um editor de código e exclua a linha que faz referência ao Jade (atenção para a vírgula após a referência do Express, ela também deve ser removida), conforme apresentado na **Listagem 5**.

E agora serão removidos os velhos templates gerados pelo Jade, acesse o diretório views no diretório do projeto e exclua todos os arquivos dentro dele. Note que os arquivos do Jade ainda estão instalados no diretório node\_modules, e estes arquivos passaram a ser inúteis no projeto gerado; para remover os arquivos da dependência acesse o terminal e execute o código:

```
npm uninstall jade
```

E substituiremos o Jade pelo EJS, para isto é necessário alterar novamente o arquivo packages.json com a dependência do EJS, na **Listagem 6** está descrito o trecho referente as dependências do sistema.

Listagem 5. Removendo o Jade da lista de dependências

```
...
"dependencies":{
  "express": "3.4.4"
}
...
```

Listagem 6. Declarando o EJS na lista de dependências

```
...
"dependencies":{
  "express": "3.4.4",
  "ejs": "*"
}
...
```

Novamente será executado o código para que o NPM realize a instalação do EJS:

```
npm install
```

Agora que tudo já está pronto para se iniciar a construção do aplicativo é necessário remover alguns trechos de código do arquivo app.js que foram automaticamente gerados pelo Express, alguns deles não são interessantes no estágio inicial de desenvolvimento e podem atrapalhar no entendimento da aplicação. Na **Listagem 7** o arquivo app.js foi editado de forma que fossem eliminados alguns recursos que não serão utilizados nos estágios iniciais da aplicação.

E antes de iniciar deve-se criar o template da página index, logo que o arquivo index.jade foi removido posteriormente. Dentro do diretório views crie o arquivo index.ejs similar ao conteúdo apresentado na **Listagem 8**.

**Listagem 7.** Adicionando suporte ao EJS no arquivo app.js.

```
var express = require('express');
var routes = require('./routes');
var http = require('http');
var path = require('path');

var app = express();

// all environments
app.set('port', process.env.PORT || 3000);
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
app.use(app.router);
app.use(express.static(path.join(__dirname, 'public')));

app.get('/', routes.index);

http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

**Listagem 8.** Template da página de boas vindas.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='utf-8' />
    <title>Nwitter - Twitter com NodeJS</title>
  </head>
  <body>
    <h1>Seja bem-vindo</h1>
  </body>
</html>
```

Precisa-se neste estágio realizar um teste para verificar se todos os elementos foram instalados e se o ambiente está configurado corretamente, basta executar o seguinte comando no terminal (lembre-se de sempre finalizar o último script que estava em execução pressionando *ctrl + c* (caso você não tenha muita experiência com o terminal do seu sistema operacional, leia a **BOX 1** para entender melhor sobre a finalização dos seus scripts):

```
node app.js
```

**BOX 1.** Finalizando scripts da forma correta

É possível que algumas vezes o desenvolvedor entre um teste de script e outro finalize a execução do script acidentalmente com o comando *ctrl + z*, isto na verdade faz com que o script continue em execução mas o usuário pode continuar sua interação no terminal, e ao tentar novamente executar o script haja um conflito de porta do servidor. Caso isto venha a acontecer o usuário esteja em um ambiente Linux ele deve executar o comando *fg* para suspender qualquer processo que esteja em execução em segundo plano.

A saída esperada deverá ser igual à **Figura 5**.

Para a próxima iteração neste projeto será criada uma tela de login do usuário para melhor entendimento da dinâmica de desenvolvimento do NodeJS e para que o projeto comece a ser organizado. Logo é necessário primeiramente criar um controller para a entidade usuário, portanto, crie um arquivo chamado *usuario.js* dentro do diretório */routes/* com o seguinte conteúdo:

```
exports.login = function(req, res){
  res.render('usuario/login');
}
```

Ou seja, foi declarada uma função *login* que por sua vez irá trazer a view *login* dentro do diretório */views/usuario/*, esta view será criada posteriormente. Mas antes é necessário contemplar esta rota dentro do arquivo *routes* para que o sistema entenda que quando o usuário fizer uma requisição na URL */usuario/login* a resposta deverá ser uma página com um formulário de login, conforme a **Listagem 9**, o arquivo *app.js* deve conter as novas rotas que a aplicação suportará.



**Figura 5.** Saída esperada da primeira iteração do projeto

**Listagem 9.** Modificando as rotas da aplicação.

```
...
var routes = require('./routes');
routes.usuario = require('./routes/usuario');
...
app.get('/', routes.index);
app.get('/usuario/login', routes.usuario.login);
...
```

Pode-se perceber que foi utilizado o próprio objeto *routes* para armazenar as rotas de usuário e que posteriormente o objeto *usuario* com todas as rotas irá chamar o método *login* ao ser requisitada a URL */usuario/login*. Como próximo passo é necessário criar a view da página de login. Para uma melhor organização dos templates, dentro do diretório */views/* crie um diretório chamado *usuario* e dentro dele crie um arquivo *ejs* chamado *login*. Para também evitar a repetição em todos os templates do sistema cabeçalho e o rodapé, deve-se separá-los em arquivos distintos e realizar um *include*. Para isso crie um arquivo *header.ejs* na raiz do diretório */views/* conforme código demonstrado na **Listagem 10**.

Caso deseje incorporar uma folha de estilo CSS, um JavaScript ou uma imagem, a inclusão é feita normalmente e a URL deve ser igual ao do diretório *public*, mas desconsidere do path o diretório *public*, por exemplo:

```
<link rel="stylesheet" type="text/css" href="/stylesheets/meu_style.css" media="all" />
```

E em seguida deve-se criar um arquivo chamado *footer.ejs* com o mesmo conteúdo apresentado na **Listagem 11** e no mesmo local onde foi salvo o arquivo *header.ejs*.

Porém, o arquivo *index.ejs* ainda contém o cabeçalho e o rodapé, portanto, deve-se remover o cabeçalho e o rodapé e inserir as notações de importação dos templates, o arquivo deve ficar desta forma:

```
<% include header.ejs %>
<h1>Seja bem-vindo!</h1>
<% include footer.ejs %>
```

E neste momento será criado o template da página de autenticação do sistema, um formulário simples, preencha o arquivo *login.ejs* com o mesmo código da **Listagem 12** que se encontra no diretório */usuario/*.

#### Listagem 10. Modificando as rotas da aplicação.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='utf-8' />
    <title>Nwitter - Twitter com NodeJS</title>
  </head>
  <body>
```

#### Listagem 11. Arquivo de rodapé da aplicação.

```
<footer>
  <p>Nwitter – Web Magazine &reg;</p>
</footer>
</body>
</html>
```

#### Listagem 12. Interface do formulário de login.

```
<% include ..header.ejs %>

<form action='/usuario/login/processa' method='POST'>
  <input type='text' name='login' placeholder='login' />
  <input type='password' name='senha' placeholder='senha' />

  <a href='/usuario/cadastro'>Criar nova conta</a>

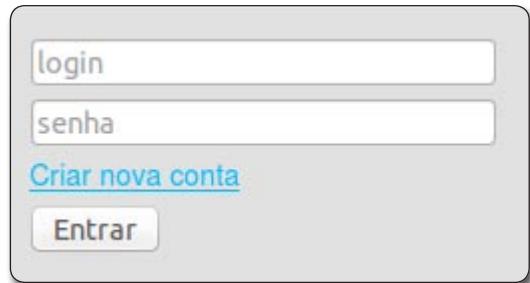
  <input type='submit' value='Entrar' />
</form>

<% include ..footer.ejs %>
```

Pode-se perceber que este formulário já conta com um link de cadastro, que por sua vez será implementado posteriormente. Antes de prosseguir, realize o teste, finalize a aplicação (caso ainda esteja em execução no terminal) e inicie novamente através do comando *node app.js* e acesse a URL */usuario/login*. O resultado apresentado deve ser similar ao formulário apresentado na **Figura 6**.

Mas ao submeter os dados no formulário é apresentado um erro “Cannot POST /usuário/login/processa” pois o processamento destes dados ainda não foi implantado. Ou seja, deve-se criar uma nova rota (neste caso, para a URL */usuario/login/processa*) e também

será necessário criar uma seção para o usuário quando o mesmo se autenticar (não será feita a implementação com o banco de dados por enquanto, apenas uma verificação se os campos foram preenchidos). Seguindo a **Listagem 13** altere o arquivo *app.js*.



**Figura 6.** Tela de login do sistema

#### Listagem 13. Adicionando suporte a cookies e inserindo a rota de processamento do login.

```
...
app.use(express.static(path.join(__dirname, 'public')));
app.use(express.cookieParser());
app.use(express.session({secret: 'seuSegredoAqui'}));
app.use(app.router);

app.get('/', routes.index);
app.get('/usuario/login', routes.usuario.login);
app.post('/usuario/login/processa', routes.usuario.login.processa);
...
```

Nesta iteração, foi criada uma chamada para dois módulos nativos do Express, o session (que é exatamente para o manuseio de sessões) e o CookieParser, que é obrigatório para que o session funcione e é também obrigatória a declaração do Cookieparser antes da declaração do session. Também é possível perceber que foi cadastrado um atributo secret dentro da chamada do session, o desenvolvedor pode preencher com qualquer string este atributo para que seja gerado um hash específico para a aplicação o que, por sua vez, agrupa mais segurança na criptografia de valores da sessão. E por fim foi relacionada a rota */usuario/login/processa* com o método processa das rotas usuário, vale ressaltar que esta URL não foi criada como resposta a um método GET mas sim utilizando POST, diferente de todas as outras que já foram registradas.

E neste momento, conforme apresentado na **Listagem 14** referente ao conteúdo do arquivo *routes.js*, será criada a função para capturar os dados que foram enviados, verificar se realmente os dados foram preenchidos e criar uma sessão de usuário, mas não é correto realizar este processamento no arquivo de rotas de usuário, esta função deve estar alocada por enquanto no router da entidade usuário, mais adiante será agregado ao projeto mais uma ferramenta para facilitar o uso dos controladores.

O código adicionado representa a criação de um novo método chamado processa (que neste caso foi adicionado dentro de login apenas para uma melhor organização do código de chamada no script *app.js*) e incluso um módulo de query string, onde os dados

recebido serão tratados e uma variável para o armazenamento dos dados recebidos, logo em seguida é inserido um callback na função on data, esta função faz o armazenamento das informações trazidas pelo form, porém, estas informações vêm na forma de query strings, por isso, elas serão tratadas na função disparada após o fim da recepção da requisição. Em seguida, ao fim da requisição (função on end) os dados recebidos são convertidos para o formato JSON e partir dai podem ser processados. Neste caso, foi feita uma verificação extremamente simples comparando se os dados recebidos estão preenchidos ou não. Caso estejam preenchidos o usuário tem sua sessão criada e é redirecionado para home page do sistema, do contrário ele é revertido à página de login sem nenhuma mensagem de erro.

**Listagem 14.** Função de validação de login e senha.

```
exports.login = function(req, res){
  res.render('usuario/login');
};

exports.login.processa = function(req, res){
  var querystring = require('querystring');
  var data = "";

  req.on("data", function(chunk) {
    data += chunk;
  });

  req.on("end", function() {
    json = querystring.parse(data);

    if((json.login != "") && (json.senha != "")){
      req.session.usuario = json.login;
      res.redirect('/');
    }else{
      res.redirect('/usuario/login');
    }
  });
};
```

O importante deste exemplo é demonstrar como o processo simples de login pode ser custoso e demorado sem o uso de duas ferramentas que podem auxiliar ainda mais o desenvolvedor e evitar que ele tenha que escrever códigos repetitivos de baixo nível. O primeiro problema que é perceptível é o código de roteamento, os scripts de routes estão responsáveis pelo processamento das requisições enquanto na verdade eles devem simplesmente armazenar as rotas e as funções do controller a chamar, sem mencionar no fato de registrar todas as rotas possíveis no arquivo app.js. Pode-se mover as notações de get e post dentro do arquivo de routes e dentro dos arquivos de rotas incluir o módulo do express, isto resolveria o problema. Porém, será adicionado ao projeto outro módulo do express que fará com que o load de todas os controllers, módulos e routes ocorra automaticamente sem que todos estes elementos sejam exaustivamente declarados no app.js, tornando a aplicação mais legível.

Na **Listagem 15** é demonstrada a adição de outro módulo ao sistema, este por sua vez, nativo do Express, é o body parser, que trará na requisição todos os campos do formulário submetido,

tornando a captação dos dados mais intuitiva e menos sujeita a falhas por parte de implementação.

Após salvar este arquivo, execute o código *npm install* no terminal. E agora será incluso o modulo de Express load e o bodyparser, altere alguns trechos do *app.js* conforme apresentado na **Listagem 16**.

**Listagem 15.** Adicionando o Express Load as dependências da aplicação.

```
...
"dependencies": {
  "express": "3.4.4",
  "ejs": "0.8.4",
  "express-load": "1.1.7"
}
...
```

**Listagem 16.** Integrando o Express Load e o Body Parser a aplicação.

```
...
var express = require('express');
var load = require('express-load');
var http = require('http');
var path = require('path');

var app = express();
...
app.use(express.cookieParser());
app.use(express.session({secret: 'webmagazine'}));
app.use(express.bodyParser());
app.use(app.router);

load('models').
then('controllers').
then('routes').
into(app);
...
```

Repare que foi movida a linha onde é declarado o app e criada uma notação para a inclusão dos módulos dos diretórios models, controllers e routers, tudo carregado dentro do objeto app. É muito importante também a ordem de carregamento destes elementos para que tudo funcione corretamente. Na **Listagem 17** estão elencadas as linhas a serem removidas do arquivo *app.js*.

**Listagem 17.** Remoção das rotas do arquivo *app.js*.

```
...
var routes = require('./routes');
routes.usuario = require('./routes/usuario');

...
app.get('/', routes.index);
app.get('/usuario/login', routes.usuario.login);
app.post('/usuario/login/processa', routes.usuario.login.processa);
...
```

Estas declarações não são mais úteis dentro do contexto do *app.js*, logo que o load das rotas também será automático. Uma vez que estas facilidades foram agregadas ao projeto, é preciso refatorar a entidade usuario. Primeiramente crie o arquivo *controllers/usuario.js*, no mesmo será inserido o processamento do login que se encontra no *routes/usuario.js*, mas também será feito uso do bodyparser.

# JavaScript Blog: Criando serviço de microblog com NodeJS

As **Listagens 18, 19 e 20** apresentam o conteúdo dos arquivos de controllers e rotas reformulados para que se faça o uso correto destes recursos.

## Listagem 18. Conteúdo do arquivo controllers/usuario.js.

```
module.exports = function(app){  
  var UsuarioController = {  
  
    login: function(req, res){  
      res.render('usuario/login');  
    },  
  
    loginAction: function(req, res){  
      if((req.body.login != "") && (req.body.senha != "")){  
        req.session.usuario = req.body.login;  
        res.redirect('/');  
      }else{  
        res.redirect('/usuario/login');  
      }  
    }  
  
    return UsuarioController;  
};
```

## Listagem 19. Reformulação do arquivo routes/usuario.js.

```
module.exports = function(app){  
  var usuario = app.controllers.usuario;  
  
  app.get("/usuario/login", usuario.login);  
  app.post("/usuario/login/processa", usuario.loginAction)  
}
```

## Listagem 20. Reformulação do arquivo routes/index.js.

```
module.exports = function(app){  
  app.get("/", function(req, res){  
    res.render('index');  
  });  
}
```

A seguir, será implantada a integração do sistema com o banco de dados, no caso deste artigo, será utilizado o banco de dados no-sql MongoDB, por sua fácil integração com o NodeJS.

## Integração com MongoDB

O MongoDB é uma base dados orientada a documentos e nos últimos anos vem sendo muito utilizada pelos desenvolvedores se destacando como o banco de dados No SQL mais utilizado no mercado, a classificação No SQL, de forma que as bases de dados são estruturadas em documentos com uma certa similaridade com a notação JSON (no caso esta formatação é na verdade chamada de BSON). Sua instalação e implantação se dão de forma fácil e intuitiva e a integração com NodeJS exige menos esforço que um banco de dados MySQL ou PostgreSQL.

Para instalar o mongoDB basta realizar o download no site oficial (veja seção **Links**), ou através do gerenciador de pacotes, no caso do Linux, e instalá-lo normalmente. Após a instalação, como demonstrado na **Listagem 21**, é necessário incluí-lo no package do projeto.

E após a alteração execute o instalador do NPM (*npm install*), e por fim se inicia a integração com a base de dados MongoDB. Na **Listagem 22** o mongoose é instanciado no arquivo principal do projeto (*app.js*).

## Listagem 21. Inclusão do mongoose nas dependências do projeto.

```
...  
"dependencies":{  
  "express":"3.4.4",  
  "ejs":"0.8.4",  
  "express-load":"1.1.7",  
  "mongoose":"3.8.0"  
}  
...
```

## Listagem 22. Carregando o mongoose no arquivo app.js.

```
...  
var express = require('express');  
var load = require('express-load');  
var http = require('http');  
var path = require('path');  
var mongoose = require('mongoose');  
...
```

Agora o sistema já suporta a interação dos models com um banco de dados MongoDB. Será criado um arquivo como função de modelo do sistema, onde são declarados os Schemas (que de certa forma se assemelham às tabelas dos bancos de dados relacionais), os campos referentes ao Schema e os relacionamentos com outros modelos. Portanto, crie um arquivo *usuario.js* dentro do diretório */models/* com o conteúdo apresentado na **Listagem 23**.

## Listagem 23. Criação do modelo de usuário.

```
module.exports = function(){  
  var Schema = db.Schema;  
  
  var usuario = Schema({  
    nome: {type: String, required: true},  
    nickname: {type: String, required: true, index:{unique:true}},  
    email: {type: String, required: true, index:{unique:true}},  
    senha: {type: String, required: true}  
  });  
  
  return db.model('usuarios', usuario);  
}
```

Ou seja, a entidade usuário possui quatro campos: nome, nickname, email e senha, todos os campos são obrigatórios e do tipo String. Não é necessário declarar um campo de ID porque o próprio mongodb se encarrega pela criação deste campo automaticamente.

Na **Listagem 24** é demonstrada a criação de um arquivo no diretório */views/usuario/* com o nome de *cadastro.ejs*, referente ao formulário de cadastro que caracteriza onde será criada a funcionalidade de login completa.

Atente ao fato também de que neste formulário de cadastro também será necessário um sistema de validação dos campos,

esta validação será realizada no controller referente à entidade do usuário, antes de implantar os métodos do controlador, na **Listagem 25** é acrescentado suporte ao validador de formulários a ser utilizado.

#### **Listagem 24.** Template do formulário de cadastro.

```
<% include ../header.ejs %>
<form action='/usuario/cadastro' method='POST'>
  <label for='usuario_nome'>Nome:</label>
  <input type='text' name='usuario[nome]' id='usuario_nome' />
  <label for='usuario_nickname'>Nickname: @</label>
  <input type='text' name='usuario[nickname]' id='usuario_nickname' />
  <label for='usuario_email'>E-mail:</label>
  <input type='text' name='usuario[email]' id='usuario_email' />
  <label for='usuario_senha'>Senha:</label>
  <input type='password' name='usuario[senha]' id='usuario_senha' />
  <label for='usuario_conf_senha'>Confirmação de senha:</label><input type='password' name='usuario[conf_senha]' id='usuario_conf_senha' />

  <input type='submit' value='Entrar' />
</form>

<% include ../footer.ejs %>
```

#### **Listagem 25.** Adicionando suporte ao Express Validator.

```
var express = require('express');
var load = require('express-load');
var expressValidator = require("express-validator");
var http = require('http');
var path = require('path');
var mongoose = require('mongoose');

...
app.set('port', process.env.PORT || 3000);
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
app.use(expressValidator);
app.use(express.static(path.join(__dirname, 'public')));
app.use(express.cookieParser());
```

O Express validator é um modulo criado como uma cópia do node-validator, mas adaptado ao express, mas todas as funções de validação são baseadas em seu módulo original, prosseguindo para o controller usuário, tornando a função de cadastro similar ao apresentado na **Listagem 26**.

Quanto ao método cadastro, não há nenhuma diferença do método login, para a renderização da página desejada, salvo a diferença do parâmetro que indica qual o template a ser renderizado. Mas o método cadastroAction apresenta uma série de novos elementos que até agora não foram contemplados. Logo nota-se o uso de uma notação assert, este conjunto de funções indica as regras dos dados submetidos pelo formulário, alimentado por dois parâmetros.

O primeiro parâmetro indica o campo a ser validado e neste caso em específico se trata de um vetor JSON, pois os campos usam o seguinte padrão de nomenclatura: *entidade[nome do campo]* para uma melhor organização dos dados recebidos pelo formulário e uma inclusão mais fácil no modelo da entidade.

O segundo parâmetro se trata da mensagem de erro a ser exibida caso o campo esteja inválido, e em seguida ao método assert, são

passados mais alguns métodos com as regras de validação (caso queira ver a lista completa, veja na seção de **Links**).

Em seguida é criada uma variável contendo todos os possíveis erros encontrados no formulário pelo módulo de validação, se houver erros a página de cadastro é renderizada novamente, mas com os erros de validação. Caso contrário, se todos os dados estiverem válidos é realizada a inclusão do cadastro do usuário. O modelo da entidade é invocado, e então este mesmo objeto realiza a criação do usuário utilizando apenas como entrada os campos do formulário de cadastro, dispensando qualquer tipo de tratamento. No call-back da função de inserção é verificado se foi encontrado algum erro de inserção, caso haja algum erro durante a inserção dos dados no MongoDB, o formulário será renderizado desta vez com os erros específicos retornados pelo próprio banco de dados. Do contrário, ele será redirecionado para a página de login.

#### **Listagem 26.** Processando entradas do usuário antes de realizar o cadastro.

```
...
cadastro: function(req, res){
  res.render("usuario/cadastro");
},
cadastroAction: function(req, res){

  req.assert(['usuario', 'name'], 'Insira seu nome completo').notEmpty();
  req.assert(['usuario', 'nickname'], 'Insira um apelido').notEmpty();
  req.assert(['usuario', 'email'], 'Insira uma conta de e-mail válida').len(10, 50)
    .isEmail();
  req.assert(['usuario', 'senha'], 'Insira uma senha de no mínimo 6 caracteres')
    .len(6, 20);
  req.assert(['usuario', 'conf_senha'], 'Confira sua senha').len(6,20);
  req.assert(['usuario', 'conf_senha'], 'As senhas não são compatíveis')
    .equals(req.body.usuario.senha);

  var errors = req.validationErrors();

  if(errors){
    res.render("usuario/cadastro", {errors: errors});
  }else{
    var usuarioModel = app.models.usuario;
    usuarioModel.create(req.body.usuario, function(error, usuario){
      if(error){
        res.render("usuario/cadastro", {errors: [{"msg": error.err}]});
      }else{
        res.redirect("/usuario/login");
      }
    });
  }
}
```

Na **Listagem 27** é apresentado o código a ser implementado para que seja criado o template que fará a apresentação de todos os erros de validação do formulário, este padrão pode ser atrelado a qualquer formulário do site que possua uma rotina de validação do Express Validator.

Conforme apresentado na **Listagem 28**, o arquivo */views/usuario/cadastro.ejs*, é alterado para que o template de erros recentemente criado seja invocado.

# JavaScript Blog: Criando serviço de microblog com NodeJS

**Listagem 27.** Template do elemento que apresentará todos os erros do formulário.

```
<% if(typeof(errors) != "undefined"){ %>
<ul>
<% for(var error in errors){ %>
<li><%= errors[error].msg %></li>
<% }
%>
</ul>
<% } %>
```

**Listagem 28.** Inclusão do modulo de erros.

```
<% include ../header.ejs %>

<% include ../errorlist.ejs %>

<form action='/usuario/cadastro' method='POST'>
...

```

Este arquivo será mantido na raiz do diretório views pois ele será utilizado em outras páginas do sistema, tal como no login, onde deverá ser incluída também a validação dos campos de nickname e senha, conforme apresentado na **Listagem 29**. Para isto no arquivo *login.ejs* dentro do diretório *views/usuario*, insira também o include para o errorlist logo após a inclusão do header. Também será necessário alterar o nome dos campos para se respeitar o mesmo padrão que na página de cadastro.

**Listagem 29.** Inclusão do modulo de erros na página de login e alteração do nome dos campos.

```
<% include ../header.ejs %>

<% include ../errorlist.ejs %>

<form action='/usuario/login' method='POST'>
<input type='text' name='usuario[nickname]' placeholder='Nickname' />
<input type='password' name='usuario[senha]' placeholder='Senha' />
```

E conforme a **Listagem 30**, também será implantado o sistema completo de login, incluindo as regras de validação.

A alteração no método consiste na inclusão de regras de validação, semelhante ao cadastro e em seguida, é feita uma busca pelos dados inseridos na base de dados, caso sejam encontrados os dados de acesso o usuário é redirecionado para a página principal do site, do contrário, a página de login será renderizada com a mensagem de erro alertando sobre os dados incorretos. E antes de prosseguir o desenvolvimento deve-se criar também o método para realizar o logout do sistema como demonstrado na **Listagem 31**.

Para que o logout funcione com a URL */usuario/logout* é necessário criar esta nova rota e direcioná-la ao método recém criado.

```
app.get("/usuario/logout", usuario.logout);
```

**Listagem 30.** Action de login utilizando a integração com banco de dados.

```
...
loginAction: function(req, res){
  req.assert(['usuario','nickname'], 'Insira um apelido').notEmpty();
  req.assert(['usuario','senha'], 'Insira uma senha de no mínimo 6 caracteres')
    .len(6, 20);

  var errors = req.validationErrors();

  if(!errors){
    var usuarioModel = app.models.usuario;
    var query = {nickname: req.body.usuario.nickname, senha: req.body.usuario.senha};

    usuarioModel.findOne(query).select('nome email nickname')
      .exec(function(error, usuario){
        if(usuario){
          req.session.usuario = usuario;
          res.redirect('/');
        }else{
          res.render('usuario/login', {errors: [{}]} );
        }
      });
  }else{
    res.render('usuario/login', {errors: errors});
  }
},
...

```

**Listagem 31.** Método de logout.

```
...
logout: function(req, res){
  if(typeof(req.session.usuario) != "undefined"){
    req.session.destroy();
    res.redirect('/');
  }
}
...

```

E para evitar que o usuário acesse a página de acesso depois de já ter realizado o login, altere o próprio arquivo de rotas do usuário e altere estas duas rotas:

```
app.get("/usuario/login", isLoggedIn, usuario.login);
app.post("/usuario/login", isLoggedIn, usuario.loginAction);
```

Na **Listagem 32** é descrita a função que deve ser criada ao fim do mesmo arquivo de rotas do usuário que impedirá o acesso dele à página de login após ele já ter validado seu acesso.

Ou seja, sempre que forem acessadas as URLs de login, tanto via GET ou POST, obrigatoriamente antes será realizado o processamento da função LoggedIn e depois do método do controller usuário.

Na **Listagem 33** está referenciado o conteúdo completo da página principal (*/views/index.ejs*) com a listagem de postagens e o painel para o usuário realizar as postagens, mas este formulário só estará disponível caso ele esteja logado no sistema.

Você tem a opção de isolar o menu em um arquivo a parte e realizar uma inclusão dele também. Agora, vamos criar o arquivo de rotas para as postagens, crie o arquivo */routes/nweets.js* com este conteúdo.

---

**Listagem 32.** Função de redirecionamento caso o usuário já esteja validado.

```
var isLoggedIn = function(req, res, next){  
    if(typeof(req.session.usuario) != "undefined"){  
        if(req.session.usuario != ""){  
            res.redirect("/");  
        }  
    }  
    next();  
}
```

**Listagem 33.** Página principal do sistema com o formulário para postagens e lista de últimos posts.

```
<% include header.ejs %>  
  
<% if(typeof(session.usuario) == "undefined"){ %>  
<ul class='menu'>  
    <li><a href='/usuario/login'>Login</a></li>  
    <li><a href='/usuario/cadastro'>Cadastro</a></li>  
</ul>  
<% } else{ %>  
<ul class='menu'>  
    <li><a href='/usuario/logout'>Logout (@<%= session.usuario.nickname %>)</a></li>  
</ul>  
<form action="/nweet/submit" method="POST">  
    <label>Insira aqui seu Nweet (140 caracteres)!</label>  
    <textarea name='nweet[texto]' id='nweet_texto'></textarea>  
    <input type='submit' value='Nweet!' />  
</form>  
<% } %>  
  
<% if(typeof(nweets) != "undefined"){ %>  
    //Lista de Nweets  
<% } %>  
  
<% include footer.ejs %>
```

As postagens também precisam de um arquivo de rotas para que a action do formulário de postagens funcione corretamente, e como apresentado na **Listagem 34**,

E na **Listagem 35** está descrita a função para o processamento das postagens do usuário.

Este controller será responsável por primeiro validar o post, se ele possui entre 1 e 140 caracteres, caso não for detectado nenhum erro, o post é criado na base de dados, mas antes é inserido o valor do campo autor com o id do usuário, previamente armazenado nas variáveis de sessão, e logo após ocorre o redirecionamento para a página principal, exibindo os últimos posts novamente. Na **Listagem 36** é apresentado o arquivo de rotas */routes/index.js* onde é criada uma rota para que a página principal receba todos os posts e os dados da sessão do usuário.

De forma que o modelo de posts faz a seleção de todos os posts por ordem de data de forma decrescente com limite 30, e após isto é renderizada a página principal atrelada também com as variáveis de sessão. E para finalizar a implantação, na **Listagem 37** é demonstrado o loop para a exibição de cada um dos posts, portanto, onde está o comentário “Lista de Nweets”.

---

**Listagem 34.** Rota da ação de postagem.

```
module.exports = function(app){  
    var nweet = app.controllers.nweet;  
    app.post("/nweet/submit", nweet.submit);  
}
```

---

**Listagem 35.** Método responsável pela criação das postagens.

```
module.exports = function(app){  
    var NweetController = {  
  
        submit: function(req,res){  
  
            req.assert(['nweet','texto'], 'Insira o conteúdo do seu nweet').len(1,140);  
  
            var errors = req.validationErrors();  
  
            if(errors){  
                res.render("/", {'errors': errors});  
            }else{  
                var nweetModel = app.models.nweet;  
  
                req.body.nweet.autor = req.session.usuario._id;  
  
                nweetModel.create(req.body.nweet, function(error, nweet){  
                    res.redirect("/");  
                });  
            }  
        }  
    };  
  
    return NweetController;  
};
```

---

**Listagem 36.** Rota para trazer todas últimas 30 postagens.

```
module.exports = function(app){  
    app.get("/", function(req, res){  
        var nweetModel = app.models.nweet;  
  
        nweetModel.find().populate('autor').sort( [['data','descending']] ).limit(30).exec(function( error, nweets ){  
            res.render('index', {'session': req.session, "nweets": nweets});  
        });  
    });  
}
```

---

**Listagem 37.** Template da listagem de todas postagens.

```
<ul>  
    <% for(var index in nweets){ %>  
        <%  
            if(nweets[index].autor != null ){  
                var dataFormatada = new Date(nweets[index].data);  
            }  
        <li>  
            <p>@<%= nweets[index].autor.nickname %>: <%= nweets[index].texto %></p>  
            <p class='publish_date'><%=  
                dataFormatada.getHours() + ":" +  
                dataFormatada.getMinutes() + ":" +  
                dataFormatada.getDate() + "/" +  
                (dataFormatada.getMonth() + 1 )  
            %></p>  
        </li>  
        <% } %>  
    <%  
    }  
    <%>  
</ul>
```

# JavaScript Blog: Criando serviço de microblog com NodeJS

Por fim, inicie a aplicação, crie um cadastro para seu usuário e crie algumas postagens. Também é possível fazer algumas melhorias, como: Adicionar suporte a upload de imagens, sistema de chat e dentre vários outros recursos para tornar o sistema mais atraente.

A aplicação demonstrada usa apenas um pequeno potencial do NodeJS, esta plataforma pode se estender nos mais diversos protocolos de comunicação, sendo uma verdadeira carta na manga no caso de alguma dificuldade em um projeto de sistemas que exija algum módulo que precise de mais agilidade no atendimento de requisições ou um volume alto de acessos dos usuários.

Alguns desenvolvedores tem inclusive portado o NodeJS para sistemas Raspberry Pi, criando um servidor de internet de extremo baixo custo e de implementação simples.

## Autor



Rafael Milléo Carrenho

[rafael.milleo@gmail.com](mailto:rafael.milleo@gmail.com) (Twitter: @milleo)



Desenvolvedor web com experiência há 5 anos no mercado, com experiência em agências e grandes empresas. Hoje prestando serviços no Portal R7 com PHP e Wordpress. Além de paralelamente exercer trabalhos de pesquisas científicas voltadas a sistemas hipermídia e Interação humano computador pela Universidade Presbiteriana Mackenzie.

## Links:

### Site oficial do NodeJS.

<http://nodejs.org/>

### Comunidade Brasileira de NodeJS.

<http://nodebr.com/>

### Informações de todos os pacotes para a plataforma NodeJS.

<https://npmjs.org/>

### Como instalar o NodeJS no Linux Ubuntu

<https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager>

### Mais informações sobre package.json

<https://npmjs.org/doc/json.html>

### Site oficial de mongodb

<http://mongodb.com>

### Site oficial do Express Validator

<https://github.com/ctavan/express-validator>

### Lista de referência de métodos de validação suportados pelo Express Validator

<https://github.com/chriso/node-validator#list-of-validation-methods>

## Conhecimento faz diferença!

The advertisement features two magazine covers. The left cover is for 'Edição 29 :: Ano 2' and the right cover is for 'Edição 28 :: Ano 2'. Both covers have the title 'engenharia de software magazine' and the DevMedia logo. The left cover has a green background and discusses 'Processo: Medição de Software: Um importante passo para a melhoria contínua'. The right cover has a blue background and discusses 'Evolução do Software: Definições, preocupações e custo'. A large red starburst graphic on the right side contains the text '+ de 290 vídeos para assinantes'.

Faça já sua assinatura digital! | [www.devmedia.com.br/es](http://www.devmedia.com.br/es)

## Faça um upgrade em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software. Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviços); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. Assine Já!



DEV MEDIA

# NodeJS: Criando uma Rede Social

## Desenvolvendo uma mini rede em tempo real

Nos últimos anos, o NodeJS vem se tornando bastante popular entre programadores, entusiastas e empresas, e mesmo ainda estando em sua fase beta, há um grande esforço de seus desenvolvedores para mantê-lo o mais estável possível.

Essa estabilidade tem motivado pequenas, médias e grandes empresas a se aventurarem a criar projetos com NodeJS em ambiente de produção.

Um dos motivos pelo qual o NodeJS tem se tornado popular é o fato de utilizar JavaScript como linguagem de programação, uma vez que praticamente todo desenvolvedor web conhece ao menos os conceitos básicos e a sintaxe do JavaScript.

Além do mais, já era um sonho antigo dos entusiastas da linguagem poder trabalhar com JavaScript no lado do servidor. Inclusive, algumas outras tentativas de implementá-lo como linguagem server side no decorrer da história da web aconteceram, porém não acabaram se popularizando tanto.

Tão logo o NodeJS começou a ficar conhecido, começaram a surgir implementações de frameworks comumente implementados em outras linguagens, como web servers, ferramentas de automatização e ORMs, multiplicando sua popularidade a cada dia.

Embora o NodeJS tenha atingido um volume considerável de usuários, mantendo um grau confiável de estabilidade e com uma curva de aprendizagem razoavelmente pequena por utilizar JavaScript como linguagem de programação, sua característica mais importante é ser baseado em entrada e saída de dados não bloqueante, também chamado de *non blocking I/O* ou ainda I/O não bloqueante. Esta última expressão será a que usaremos para esse artigo.

Isso quer dizer que, diferente do comportamento tradicional das tecnologias de programação, as requisições feitas ao NodeJS que envolvem entrada ou saída de dados não permanecem presas ao processo até a sua conclusão.

Ao invés disso, são utilizadas solicitações com funções de callback, que não ficam presas ao processo. Quando o programa estiver pronto para entregar o resultado da

### Fique por dentro

Neste artigo falaremos do aspecto mais importante do NodeJS, e que foi fundamental para a sua propagação: a programação orientada a eventos. Veremos também uma das suas principais características em comparação com outras tecnologias semelhantes: a forma como ele lida com I/O não bloqueante.

Após isso, teremos a oportunidade de ver um exemplo de construção de uma rede social que simule bem todos os aspectos mais importantes do NodeJS, desde sua instalação e configuração, até conceitos mais aprofundados como a construção das camadas cliente e servidor e a API Socket.IO.

solicitação ele fará uso destas funções de callback, entregando o conteúdo solicitado como parâmetro das mesmas.

Essa programação orientada a eventos assíncronos é o que faz com que o NodeJS tenha a característica de criar aplicações em tempo real.

### Aplicação exemplo

Para demonstrar como o NodeJS lida com aplicações em tempo real através da programação orientada a eventos e I/O não bloqueante, vamos apresentar uma aplicação web completa.

A aplicação é uma mini rede social volátil, oportunamente chamada de RAMBook, onde será possível criar posts, comentá-los e curtí-los em tempo real, porém, cujos dados não serão persistidos em uma base de dados ou sistema de arquivos.

Devido a essa característica de não persistir os dados, qualquer usuário poderá se conectar à mini rede social sem precisar se autenticar, e sempre que um novo usuário entrar, será exibida a sua timeline vazia, um campo de texto para criar novos posts e a lista de usuários conectados, como apresentado na **Figura 1**.

### Ingredientes da aplicação

Como deixamos a stack de persistência de dados de fora para este exemplo, ficamos com APIs da camada de apresentação e as regras do lado do servidor apenas.

Vale esclarecer que alguns códigos não estão necessariamente utilizando melhores práticas de desenvolvimento, em prol da didática.

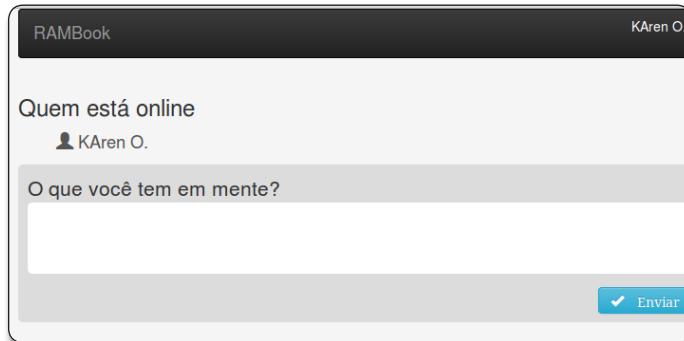


Figura 1. Tela da mini rede social após o login do usuário

Na camada cliente, serão utilizadas as seguintes ferramentas:

- Twitter Bootstrap;
- RequireJS;
- MustacheJS;
- jQuery;
- Bower.

Na camada server do NodeJS utilizaremos:

- Socket.IO;
- ExpressJS;
- Nodemon.

## Preparação do ambiente de desenvolvimento

O primeiro e mais óbvio passo é instalar o NodeJS, e uma vez que estiver instalado, vamos utilizar o NPM, o gerenciador de pacotes do NodeJS, para instalar as demais dependências do projeto.

Se você ainda não tiver o NodeJS instalado em sua máquina, ou se possui uma versão muito antiga, acesse o link de download do mesmo disponível na seção **Links** e efetue o download para o seu sistema operacional, realizando também a sua instalação.

Se for preciso, faça o mapeamento do diretório bin nas variáveis de ambiente do seu sistema operacional de maneira a poder acessar os binários node e npm de qualquer lugar.

## Instalando as dependências

Uma vez que o NodeJS estiver devidamente instalado e configurado, é hora de instalar as dependências do projeto através do NPM, o Node Packaged Modules.

O NPM é um repositório de módulos para NodeJS repleto de APIs que podem ser facilmente baixadas, muito semelhante ao apt-get do Linux Debian, ou o Gem do Ruby.

Para nossa stack de backend vamos instalar o Socket.IO e o ExpressJS, e para as dependências da camada de front-end vamos utilizar o módulo Bower, que funciona de uma maneira extremamente semelhante ao próprio NPM. Porém o Bower gerencia APIs de JavaScript como jQuery e BackboneJS.

Existem outros módulos muito bons para a criação de projetos web como o Grunt, Gulp e Yeoman, que trabalham com geradores e utilizam o Bower e Grunt, colocando-os para trabalhar lado a lado de uma forma mais automatizada.

Porém, para o nosso exemplo a HTML, CSS e código JavaScript serão escritos “do zero”. Contudo, se você se sente confortável em trabalhar com ferramentas automatizadas, fique à vontade para utilizar o que preferir.

### Nota

Todos os links para download das ferramentas citadas encontram-se disponíveis na seção **Links**.

## Baixando as dependências da camada server

Crie um diretório chamado rambook e acesse-o para iniciar o download das dependências.

Vamos começar pelas dependências da camada server, utilizando o comando **npm install** conforme mostrado na **Listagem 1**.

### Listagem 1. Baixando as dependências do backend.

```
npm install Socket.IO  
npm install express  
npm install nodemon -g  
npm install bower -g
```

Note que para o download dos módulos Nodemon e Bower utilizamos o parâmetro **-g**, que é responsável por efetuar o download dos módulos e os instalar em um diretório de módulos globais, junto ao diretório onde você instalou o NodeJS.

Ambos os módulos possuem arquivos binários, e podem ser executados como qualquer arquivo binário do seu sistema operacional.

Como já havia sido explicado anteriormente, o Bower funciona como um gerenciador de pacotes para APIs JavaScript, muito semelhante ao NPM.

O Nodemon é um “watcher”, que fica “escutando” quando qualquer arquivo dentro do diretório for alterado, e faz um “refresh” na aplicação NodeJS quando isso acontece. É uma ferramenta que auxilia durante o desenvolvimento da aplicação, porém não é necessário (nem recomendado) sua utilização em ambiente de produção.

Há outras opções interessantes que podem ser usadas com o NPM, como preservar a referência das dependências em um arquivo package.json com a opção **--save**, e assim poder distribuir a aplicação apenas com os códigos fonte em controladores de versão como o Github, sem precisar fazer upload das dependências. Assim, outros desenvolvedores que fizerem download da aplicação precisarão apenas digitar o comando **npm install** e o NPM irá ler as dependências do arquivo package.json.

## Baixando as dependências da camada client

Dentro do diretório rambook, crie um diretório chamado web e acesse esse diretório, pois é dentro dele que iremos baixar as dependências JavaScript/CSS, executando os comandos da **Listagem 2**.

Ainda no diretório web, crie os diretórios scripts e styles, onde ficarão nossos arquivos CSS e JavaScript da aplicação na stack de client.

Após o término dos downloads, você pode notar um novo diretório chamado bower\_components que possui uma estrutura de diretório para cada API instalada pelo Bower.

Por default, o Bower busca as APIs no Github, mas você pode especificar parâmetros para acessar outras fontes.

Além disso, assim como o NPM, o Bower possui um parâmetro --save que persiste a referência das dependências em um arquivo bower.json.

## Estrutura do projeto

Até o momento temos uma estrutura de diretórios com todas as bibliotecas necessárias e prontas para trabalhar, conforme exibido na **Listagem 3**.

**Listagem 2.** Baixando as dependências da camada client.

```
bower install jquery  
bower install bootstrap  
bower install mustache  
bower install requirejs
```

**Listagem 3.** Árvore de diretórios do projeto.

```
rambook  
└── node_modules  
    ├── express  
    └── Socket.IO  
└── web  
    ├── bower_components  
    ├── scripts  
    └── styles
```

## Preparando os arquivos do servidor

Vamos utilizar o diretório “root” rambook para os arquivos da camada server, e o diretório web onde todo o conteúdo estático ficará.

No diretório rambook, crie os seguintes arquivos:

- **index.js**, que servirá como nosso “main program” e iniciará os servidores web e websocket;
- **server.js**, responsável por instanciar, configurar e exportar os servidores http e websocket;
- **userhandling.js**, onde residirá nossa regra de negócios para gerenciar as ações da mini rede social.

## Preparando os arquivos do client

No diretório rambook/web crie os seguintes arquivos:

- index.html, que será a página da nossa aplicação. Este será o único HTML, pois faremos uma SPA, Single Page Application;
- styles/main.css, que conterá o estilo de nossa aplicação
- scripts/main.js, onde estará a configuração do RequireJS e servirá como nosso “main program”;
- scripts/client.js, onde estará nossa socket client que se comunicará com o servidor websocket;
- scripts/view.js, responsável por se comunicar com a client.js e fazer o link entre a websocket e os eventos dos elementos HTML da página.

Ao término da criação dos arquivos da camada client e server, devemos ter uma estrutura como exibido na **Listagem 4**.

**Listagem 4.** Árvore de diretórios e arquivos do projeto.

```
rambook  
├── index.js  
├── node_modules  
│   └── express  
│       └── Socket.IO  
├── package.json  
├── server.js  
└── userhandling.js  
└── web  
    ├── bower_components  
    │   ├── bootstrap  
    │   ├── jquery  
    │   ├── mustache  
    │   └── requirejs  
    ├── bower.json  
    ├── index.html  
    ├── scripts  
    │   ├── client.js  
    │   ├── main.js  
    │   └── view.js  
    └── styles  
        └── main.css
```

Note que foram omitidos arquivos das dependências, como as do Socket.IO e ExpressJS.

Também foram omitidas as dependências baixadas pelo Bower, pois como ele trabalha com os repositórios das APIs do Github, muitos arquivos desnecessários para o projeto estão armazenados.

## Construindo a camada front-end da aplicação

Primeiro, vamos escrever o conteúdo de nosso HTML, que foi montado utilizando o Twitter Bootstrap.

Caso você vá digitar o conteúdo desse arquivo, sugiro que utilize algum template, como os exemplos fornecidos pelo site do Bootstrap, ou utilize uma ferramenta de automatização como o Yeoman, ou ainda, utilize uma IDE que facilite a digitação de código com macros e “auto-complete text” como o Sublime Text, por exemplo.

## Entendendo a estrutura do HTML

Se você já estiver familiarizado com o Bootstrap, não encontrará problemas para entender o código. Contudo, para facilitar a compreensão de todos, segue uma explicação de como o código HTML está organizado:

- No topo do código, temos o tradicional cabeçalho, onde estão sendo importados os estilos CSS e o tema do Bootstrap, o título da página e demais tags como a <meta>;
- A tag <nav> que marca o header da aplicação, onde reside o nome da aplicação e o formulário de login;
- A lista de usuários conectados a mini rede social apresentada na tag <ul>;
- Uma tela de boas vindas, solicitando ao usuário para que faça o login e possa então acessar o sistema;

# NodeJS: Criando uma Rede Social

- Um `<div>` oculto com o campo de texto para criar posts e a timeline do usuário;
- Tags `<script>` com o atributo `type` configurado para `text/template`, onde estão os templates a serem utilizados pelo Mustache na hora de renderizar os fragmentos da aplicação;

- E, finalmente, o include do RequireJS e seu arquivo principal.

O conteúdo de nosso arquivo HTML pode ser visto na **Listagem 5**. Note na listagem que ambos os includes de CSS e do RequireJS estão apontando para o diretório `bower_components`, e em

**Listagem 5.** Conteúdo do documento index.html

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>RAMBook</title>
    <meta name="description" content="">
    <meta name="viewport" content="width=device-width">
    <link rel="stylesheet" type="text/css" href="bower_components/bootstrap/dist/css/bootstrap.min.css"/>
    <link rel="stylesheet" type="text/css" href="bower_components/bootstrap/dist/css/bootstrap-theme.min.css"/>
    <link rel="stylesheet" href="styles/main.css">
  </head>
  <body>
    <div class="container">
      <nav class="navbar navbar-inverse" role="navigation">
        <div class="navbar-header">
          <span class="navbar-brand">RAMBook</span>
        </div>
        <div class="navbar-form navbar-right hidden" id="frm-join">
          <div class="form-group">
            <input type="text" id="username" maxlength="10" class="form-control input-sm" placeholder="Digite seu nome">
          </div>
          <button type="button" id="btn-entrar" class="btn btn-info input-sm">Entrar</button>
        </div>
        <div></div>
      </nav>

      <div class="row">
        <div class="col-md-3">
          <h3>Quem está online</h3>
          <ul class="online-users"></ul>
        </div>
        <div class="col-md-9">

          <div id="not-logged">
            <h2>Você precisa se conectar</h2>
            <h3>Digite seu nome e clique em "Entrar"</h3>
          </div>

          <div id="logged" class="hidden">
            <div class="post-form">
              <div class="action">O que você tem em mente?</div>
              <textarea class="txt" id="post-content"></textarea>
              <div class="pull-right">
                <button class="btn btn-info glyphicon glyphicon-ok" id="btn-postar">Enviar</button>
              </div>
            </div>
            <div class="posts"></div>
          </div>
        </div>
      </div>
    </div>
    <script type="text/template" id="userlist-template">
      {{#users}}
        <li data-user-id="{{id}}><span class="glyphicon glyphicon-user"></span>
        {{username}}</li>
      {{/users}}
    </script>

    <script type="text/template" id="post-template">
      {{#post}}
        <div class="post" data-post-id="{{id}}>
          <div class="poster-info">
            <span class="poster">{{author}}</span>
            <span class="action">escreveu, as </span>
            <span class="time">{{hora}} hs</span>
          </div>
          <div class="post-body">{{text}}</div>
          <div class="post-actions">
            <a class="glyphicon glyphicon-thumbs-up lnk-like-post" data-post-id="{{id}}" data-like="true">(0)</a> &ampnbsp
            <a class="glyphicon glyphicon-comment lnk-comment" data-post-id="{{id}}></a>
          </div>
        </div>
        <div data-post-to-comment-id="{{id}} class="comment-form hidden">
          <div class="action">Deixe seu comentário</div>
          <textarea class="txt"></textarea>
          <div class="pull-right">
            <button class="btn btn-info btn-sm glyphicon glyphicon-ok btn-commentar">Enviar</button>
            <button class="btn btn-info btn-sm glyphicon glyphicon-remove lnk-cancel-comment" data-post-id="{{id}}> Cancelar</button>
          </div>
        </div>
        <div class="post-comments"></div>
      {{/post}}
    </script>

    <script type="text/template" id="comment-template">
      {{#comment}}
        <div class="comment" data-comment-id="{{id}}>
          <div class="poster-info">
            <span class="poster">{{author}}</span>
            <span class="action">comentou, as </span>
            <span class="time">{{hora}} hs</span>
          </div>
          <div class="post-body">{{text}}</div>
          <div class="post-actions">
            <a class="glyphicon glyphicon-thumbs-up lnk-like-comment" data-comment-id="{{id}}" data-like="true">(0)</a> &ampnbsp
          </div>
        </div>
      {{/comment}}
    </script>

    <script src="bower_components/requirejs/require.js"
           data-main="scripts/main"></script>
  </body>
</html>
```

seguida, para os respectivos diretórios de cada biblioteca, com exceção de nosso próprio CSS, que está sendo direcionado para styles/main.css. O conteúdo do arquivo main.css pode ser visto na **Listagem 6**.

Vale destacar que, caso você decida fazer o deploy de sua aplicação com as APIs contidas no diretório bower\_components, você deve excluir os arquivos desnecessários, ou copiar as APIs para outro diretório, ou ainda, preferivelmente, utilize uma ferramenta de automatização como as tasks do Grunt para realizar essa tarefa para você.

## Importando as dependências do JavaScript

Como você pôde notar, a única dependência de JavaScript que temos é a do RequireJS, seguido de sua dependência de arquivo principal, main.js, no atributo data-main="scripts/main". O RequireJS pode ser tratado basicamente como uma API para modularização e carregamento assíncrono "on demand" de arquivos JavaScript.

O conteúdo do main.js aparece na **Listagem 7** e apresenta como os módulos de nossa aplicação na camada client estão configurados.

O código é autoexplicativo: trata-se basicamente de um objeto literal com as configurações do caminho base dos módulos, os caminhos para os módulos jQuery, Mustache e nossa websocket client, gentilmente fornecida pelo próprio Socket.IO server, como veremos mais à frente.

Ao final do código, é importado o módulo view, referente ao arquivo view.js, que pode ser analisado na **Listagem 8** e então este é executado.

Apesar de ser um pouco longo, o arquivo view.js é muito simples de ser entendido, pois ele é a ligação entre a interação do usuário e a comunicação com a websocket client, que por sua vez se comunica diretamente com o nosso servidor websocket.

No topo do arquivo, pode-se notar a importação das dependências do jQuery, usado para ler elementos DOM do nosso index.html e realizar binds das ações do usuário, o MustacheJS usado para fazer a cola dos dados recebidos do servidor e os templates

**Listagem 6.** Conteúdo do arquivo de estilos styles/main.css.

```
body {  
    background-color: rgb(245, 245, 245);  
    overflow-y: scroll;  
}  
  
.frm-join {  
    color: #fff;  
}  
  
.log {  
    font-size: 25px;  
    text-align: center;  
    color: red;  
}  
  
.post {  
    border-bottom: 1px solid #aaa;  
    margin-top: 25px;  
}  
  
.poster-info {  
    font-size: 120%;  
    margin-bottom: 5px;  
    padding: 5px;  
}  
  
.poster {  
    color: #2aab2d;  
    font-weight: bold;  
}  
  
.post-body {  
    min-height: 30px;  
    color: #555;  
    padding: 5px;  
}  
  
.post-actions {  
    text-align: left;  
    padding: 10px;  
}  
  
.post-comments {  
    border-top: 1px dotted #bbb;  
}  
  
.comment {  
    font-size: 90%;  
    padding-left: 20px;  
}  
  
.post-form, .comment-form {  
    overflow: auto;  
    padding: 10px;  
    background-color: rgb(235, 235, 235);  
    border-radius: 5px;  
}  
  
.post-form {  
    font-size: 150%;  
    background-color: rgb(220, 220, 220);  
}  
  
.txt {  
    width: 100%;  
    height: 75px;  
    border: none;  
    border-radius: 5px;  
    margin-bottom: 10px;  
}  
  
.online-users {  
    list-style-type: none;  
    font-size: 130%;  
    color: #555;  
}  
  
a {  
    text-decoration: none !important;  
    cursor: pointer;  
}
```

# NodeJS: Criando uma Rede Social

## Listagem 7. Configurações do RequireJS.

```
requirejs.config({
  basePath: '/',

  paths: {
    'jquery': './bower_components/jquery/jquery.min',
    'socketio': 'Socket.IO/Socket.IO',
    'mustache': './bower_components/mustache/mustache'
  },
  shim: {
    mustache: {
      exports: 'Mustache'
    }
  }
});
```

```
require(['view'], function(view) {});
```

## Listagem 8. Arquivo view.js.

```
require(['jquery', 'client', 'mustache'], function($, client, Mustache) {

  var username = $('#username'),
    onlineUsers = $('#online-users'),
    postContent = $('#post-content'),
    frmJoin = $('#frm-join'),
    posts = $('#posts'),
    notLogged = $('#not-logged'),
    logged = $('#logged'),
    userListTpl = $('#userlist-template').html(),
    postTpl = $('#post-template').html(),
    commentTpl = $('#comment-template').html(),
    lnkComment = $('#lnk-comment');

  //Formatador da hora para o mustache
  function formatHour() {
    var today = new Date(this.timestamp);
    return today.getHours() + ':' + today.getMinutes();
  }

  /*****DECLARAÇÃO DE EVENTOS*****/
  $('#container')

  .on('click', '#btn-entrar', function(ev) {
    var name = username.val();

    if(name) {
      client.userLogin(name);
    }
  })

  .on('click', '.lnk-comment', function(ev) {
    var postId = $(this).data('post-id');
    $('#data-post-to-comment-id=' + postId + "").toggleClass('hidden');
  })

  .on('click', '#btn-postar', function(ev) {
    client.makePost(postContent.val());
    postContent.val('');
  })

  .on('click', '.btn-commentar', function(ev) {

    var commentForm = $(this).parents('.comment-form'),
      postId = commentForm.data('post-to-comment-id'),
      textarea = $('#txt', commentForm),
      commentData = {postId: postId, text: textarea.val()};

    client.makeComment(commentData);
    textarea.val('');
    commentForm.addClass('hidden');
  })

  .on('click', '.lnk-cancel-comment', function(ev) {
    var commentForm = $(this).parents('.comment-form'),
      postId = commentForm.data('post-to-comment-id'),
      textarea = $('#txt', commentForm),
      commentData = {postId: postId, text: ''};

    client.cancelComment(commentData);
    textarea.val('');
    commentForm.removeClass('hidden');
  })
});
```

```
var commentForm = $(this).parents('.comment-form');
commentForm.addClass('hidden');
})

.on('click', '.lnk-like-post', function(ev) {
  var postId = $(this).data('post-id'),
    like = $(this).attr('data-like');

  if(like === 'true') {
    like = true;
  } else {
    like = false;
  }

  client.likePost({postId: postId, like: like});
  $(this).attr('data-like', !like);
})

.on('click', '.lnk-like-comment', function(ev) {
  var commentId = $(this).data('comment-id'),
    like = $(this).attr('data-like');

  if(like === 'true') {
    like = true;
  } else {
    like = false;
  }

  client.likeComment({commentId: commentId, like: like});
  $(this).attr('data-like', !like);
})

.*****DECLARAÇÃO DOS LISTENERS*****
client.events.connect = function(data) {
  frmJoin.removeClass('hidden');
};

client.events.userlogin = function(data) {
  frmJoin.html(username.val());

  notLogged.addClass('hidden');
  logged.removeClass('hidden');
};

client.events.userlist = function(data) {
  var html = Mustache.render(userListTpl, {users: data});
  onlineUsers.html(html);
};

client.events.makepost = function(data) {
  var html = Mustache.render(postTpl, {post: data, hora: formatHour});
  posts.prepend(html);
};
```

que vimos nas tags <script> na HTML, e, finalmente, a importação da nossa camada de negócios, client.js, que pode ser vista na **Listagem 9**.

#### Continuação: Listagem 8. Arquivo view.js.

```
client.events.makecomment = function(data) {
  var html = Mustache.render(commentTpl, {comment: data, hora: formatHour}),
    postComments = $(':post[data-post-id=' + data.postId + '].post-comments');
  postComments.prepend(html);
};

client.events.likepost = function(data) {
  var InkLikePost = $(':post[data-post-id=' + data.postId + '].lnk-like-post');
  InkLikePost.html('(' + data.numLikes + ')');
};

client.events.likecomment = function(data) {
  var InkLikeComment = $(':comment[data-comment-id=' + data.commentId + '].lnk-like-comment');
  InkLikeComment.html('(' + data.numLikes + ')');
};

});
```

Muito semelhante ao view.js, o arquivo client.js está dividido em listeners e eventos, chamados de emits pelo Socket.IO, o qual entraremos em mais detalhes a seguir.

Note que a única dependência do client.js é a API do Socket.IO, um framework responsável por se comunicar com o servidor de websocket.

### Entendendo o Socket.IO no lado do client

Fazendo uso da programação orientada a eventos do NodeJS, a proposta do Socket.IO é oferecer de forma transparente ao programador as diversas maneiras de se trabalhar com conexões persistentes entre o browser e o servidor.

A maneira mais coerente de fazer isso é através da API de Websocket, apresentada na especificação da HTML5.

A maioria dos browsers hoje em dia já possuem a implementação dessa especificação, porém, alguns browsers mais antigos não contam com a API da Websocket.

Nesses casos há diversas soluções como o polling, long polling, Flash Websocket e outros mais que tem a característica de simular de forma paliativa o comportamento de uma websocket.

Quando o programador precisa se preocupar com os vários browsers e versões, uma porção de validações precisam ser feitas para identificar qual é a solução mais adequada para lidar com conexões persistentes. E o pior, é ter que implementar todas elas de forma a atender todos os públicos.

Com o advento do combo NodeJS + Socket.IO essa dor de cabeça terminou, e o programador pode se concentrar nas regras de negócio da aplicação.

Quando o browser se conecta através da API do Socket.IO.js, este se encarrega de fazer as devidas validações, preparar o contorno mais coerente, se necessário, e se comunicar com o servidor passando todas essas informações.

#### Listagem 9. Regras de negócio da camada client – client.js.

```
define(['socketio'], function(io) {

  var socket = io.connect('/'),
    events = {};

  socket.emit('likecomment', {});
  socket.emit('unlikecomment', {});

  //listeners
  socket.on('connect', function(data) {
    events.connect(data);
  });

  socket.on('msg', function(data) {
    console.log(data);
  });

  socket.on('userlogin', function(data) {
    events.userlogin(data);
  });

  socket.on('userlist', function(data) {
    events.userlist(data);
  });

  socket.on('makepost', function(data) {
    events.makepost(data);
  });

  socket.on('makecomment', function(data) {
    events.makecomment(data);
  });

  socket.on('likepost', function(data) {
    events.likepost(data);
  });

  socket.on('likecomment', function(data) {
    events.likecomment(data);
  });

  //emits
  function userLogin(username) {
    socket.emit('userlogin', {username: username});
  }

  function makePost(post) {
    socket.emit('makepost', post);
  }

  function makeComment(comment) {
    socket.emit('makecomment', comment);
  }

  function likePost(likeData) {
    socket.emit('likepost', likeData);
  }

  function likeComment(likeData) {
    socket.emit('likecomment', likeData);
  }

  return {
    socket: socket,
    userLogin: userLogin,
    makePost: makePost,
    makeComment: makeComment,
    events: events,
    likePost: likePost,
    likeComment: likeComment
  };
});
```

# NodeJS: Criando uma Rede Social

Além do mais, caso o browser do usuário possua suporte a Websockets, uma série de passos precisam ser seguidos para conectar o usuário ao servidor, como o handshake, que é uma comunicação HTTP inicial responsável por validar e autenticar o usuário no servidor, até a conexão com o servidor da Websocket de fato.

## Aprendendo a lidar com a API do Socket.IO

Diferentemente do protocolo HTTP tradicional, quando emitimos um evento ao servidor da Websocket, não ficamos “pendurados” ao servidor até que este retorne uma resposta de conteúdo ou uma mensagem de erro.

Ao invés disso, e devido ao seu comportamento orientado a eventos assíncronos e não bloqueantes, acabamos criando nosso próprio mini protocolo de comunicação. Ou seja, por um lado, emitimos uma mensagem ao servidor, e precisamos declarar uma escuta para “ouvir” a resposta do mesmo, que pode acontecer a qualquer momento, e não imediatamente após receber nossa mensagem.

Por isso, basicamente o Socket.IO trabalha com duas funções **emit** e **on**, como pode ser visto no `client.js`. Ou seja, quando queremos emitir uma mensagem ao servidor, usamos a função `emit`, seguida de um nome de evento definido pelo programador e os dados a serem recebidos pelo servidor.

Quando queremos escutar um evento recebido do servidor, precisamos implementar a função `on`, cujos parâmetros são uma string com o nome do evento, e uma função de callback cujo parâmetro é a informação enviada pelo servidor.

Por exemplo, quando queremos saber se um usuário está logado, utilizamos o evento `userlogin`, como no exemplo abaixo:

```
socket.on('userlogin', function(data) {  
  events.userlogin(data);  
});
```

Note que estamos escutando o evento `userlogin`, e os dados recebidos pelo parâmetro `data` são enviados para o objeto `events`, que será posteriormente lido pelo `view.js`.

Por outro lado, quando queremos notificar o servidor de que um novo usuário quer se conectar, emitimos o evento com o nome `userlogin`. Utilizamos o mesmo nome para eventos que representam a mesma ação para facilitar o entendimento:

```
function userLogin(username) {  
  socket.emit('userlogin', {username: username});  
}
```

A maneira como a comunicação orientada a eventos funciona ficará mais clara quando virmos o código do servidor.

## Construindo a camada backend da aplicação

Recapitulando a estrutura de arquivos da nossa camada do servidor, possuímos três arquivos: `index.js`, `server.js` e `userhandling.js`.

O conteúdo do `index.js`, exibido na **Listagem 10**, é bastante intuitivo.

O arquivo `index.js` é muito simples, pois apenas instancia os módulos de nossa aplicação e faz a união destes, iniciando finalmente o servidor.

Primeiro, são importados os módulos em variáveis, como `socketServer` que retém a referência ao `Socket.IO`, `webServer` com a referência ao servidor Web e o objeto `userHandling`, que possui toda a regra de negócio da nossa mini rede social volátil.

Então, o servidor web é iniciado na porta 9000 e o objeto `userHandling` é linkado ao servidor da `Websocket`.

O arquivo `server.js`, que pode ser analisado na **Listagem 11**, bem como o `index.js`, faz referência aos módulos do ExpressJS e `Socket.IO`. Então finalmente os instancia e os prepara para serem utilizados pelo `index.js`.

### Listagem 10. Conteúdo do index.js.

```
//Carrega o módulo server.js  
var Server = require('./server'),  
  
//Carrega o objeto io  
socketServer = Server.socketServer,  
  
//Carrega o objeto httpServer  
webServer = Server.webServer,  
  
UserHandling = require('./userhandling'),  
  
userHandling = new UserHandling(socketServer),  
  
//Inicia o web server na porta 9000  
webServer.listen(9000);
```

### Listagem 11. Conteúdo do arquivo server.js.

```
process.title = 'TTT Server';  
  
//Importando ExpressJS  
var express = require('express'),  
  
//Importando Socket.IO  
socketio = require('Socket.IO'),  
  
//Criando uma instância do ExpressJS  
app = express(),  
  
//Criando um HTTP Server a partir do ExpressJS  
httpServer = require('http').createServer(app),  
  
//Utilizando a mesma porta do HTTP Server para o Socket.IO  
io = socketio.listen(httpServer)  
;  
  
//Diz ao Express que o diretório web contém conteúdos estáticos  
app.use(express.static(__dirname + '/web'));  
  
//Exporta os módulos  
module.exports.socketServer = io;  
module.exports.webServer = httpServer;
```

Basicamente, a maior diferença na estrutura de código do *server.js* e o *index.js* é a utilização do *process.title*, que registra o nome do processo no sistema operacional. Entretanto, essa funcionalidade pode não funcionar em todos os sistemas operacionais, mas também não afetará o funcionamento da aplicação.

Após a carga dos módulos, estes são preparados, configurados e atados uns aos outros.

Note que o servidor de websocket do Socket.IO consegue aproveitar o listener do web server do ExpressJS.

Por último, mas não menos importante, aliás, talvez o arquivo mais importante de nossa aplicação, a **Listagem 12** apresenta o conteúdo do *userhandling.js*, que retém toda a regra de negócio da mini rede social volátil.

É nele que estão todos os eventos que são emitidos e “escutados” pelo Socket.IO em relação ao client.

Embora um pouco mais comprido do que os outros arquivos da camada de backend, o arquivo *userhandling.js* se assemelha muito à sua versão da camada client.

Além dos eventos tradicionais referentes à rede social, como logar usuário, fazer post, comentários e likes, o *userhandling* possui dois eventos especiais: connection e disconnect.

O evento connection é disparado logo após o handshake, ou seja, logo após o client se comunicar com o servidor através de uma comunicação utilizando o protocolo HTTP.

Uma vez que ambos client e server “apertam as mãos” uns dos outros, o servidor permite ao client o acesso a websocket.

Ao mesmo passo em que, quando um usuário se desconecta do servidor, seja por ter fechado o browser ou ter disparado uma ação de desconexão, o evento disconnect é acionado e uma série de ações é realizada.

No nosso caso, quando o evento connection é disparado, fazemos o bind de todos os eventos que o server irá escutar do client e vice-versa, e também emitimos um evento userlist dizendo a todos os usuários conectados que um novo usuário entrou na rede social.

Da mesma maneira, o evento disconnect remove o usuário da lista de usuários online e avisa a todos os outros usuários, através do objeto socket.broadcast.emit com a nova lista de usuários online.

Outro ponto interessante é a maneira como estamos lidando com as curtidas de posts e comentários. Como não temos uma maneira de persistir os dados em disco, foi criado o objeto this.posts, como pode ser visto no começo do arquivo *userhandling.js*. Este objeto mantém um histórico do id dos posts e quantas curtidas eles receberam. Para se ter uma noção de como este controle funciona, dê uma olhada nas funções onLikeComment e onLikePost. Os ids dos posts foram feitos com o timestamp da hora do post.

## CURSOS ONLINE

A Revista SQL Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.



### CONHEÇA OS CURSOS MAIS RECENTES:

- **Cursos: Curso de noSQL (Redis) com Java**
- **Desenvolvimento para SQL Server com .NET**
- **Curso PostgreSQL - Treinamento de banco de dados (Curso Básico)**

Para mais informações :

<http://www.devmedia.com.br/cursos/banco-de-dados>

(21) 3382-5038

# NodeJS: Criando uma Rede Social

**Listagem 12.** Regras de negócio da rede social no arquivo userhandling.js.

```
var UserHandling = module.exports = function(io) {  
  
    //Mantem controle dos "likes" para cada post  
    this.posts = {};  
  
    this.io = io;  
    this.loggedUsers = {};  
    this.init();  
};  
  
UserHandling.prototype = {  
  
    init: function() {  
        var that = this;  
        this.io.on('connection', function(socket) {  
  
            socket.on('disconnect', function() {  
                that.onDisconnect(socket);  
            });  
  
            that.bindEvents(socket);  
            that.onUserList(socket);  
        });  
    },  
  
    bindEvents: function(socket) {  
        var that = this;  
  
        socket.on('userlogin', function(data) {  
            that.onUserLogin(socket, data);  
        });  
  
        socket.on('userlist', function(data) {  
            that.onUserList(socket, data);  
        });  
  
        socket.on('makepost', function(data) {  
            that.onMakePost(socket, data);  
        });  
  
        socket.on('makecomment', function(data) {  
            that.onMakeComment(socket, data);  
        });  
  
        socket.on('likepost', function(data) {  
            that.onLikePost(socket, data);  
        });  
  
        socket.on('likecomment', function(data) {  
            that.onLikeComment(socket, data);  
        });  
  
        },  
  
        onDisconnect: function(socket) {  
            delete this.loggedUsers[socket.id];  
            this.onUserList(socket);  
        },  
  
        sendMessage: function(socket, msg) {  
            socket.emit('msg', {text: msg, type: 'info'});  
        },  
  
        onUserLogin: function(socket, data) {  
            var that = this;  
            if(data.username) {  
                socket.handshake.username = data.username;  
                that.loggedUsers[socket.id] = socket;  
                socket.emit('userlogin');  
                that.onUserList(socket);  
            } else {  
                this.sendMessage(socket, 'vc precisa escolher um nome');  
            }  
        },  
  
        onUserList: function(socket, data) {  
            var that = this,  
                keys = Object.keys(this.loggedUsers),  
                userList = new Array(keys.length),  
                i = 0;  
  
            keys.forEach(function(k) {  
                userList[i++] = {  
                    id: k,  
                    username: that.loggedUsers[k].handshake.username  
                };  
            });  
  
            socket.emit('userlist', userList);  
            socket.broadcast.emit('userlist', userList);  
        },  
  
        onMakePost: function(socket, data) {  
            var id = Date.now(),  
                postData = {  
                    author: socket.handshake.username,  
                    authorId: socket.id,  
                    timestamp: id,  
                    id: id,  
                    text: data  
                };  
  
            //cadastra o post no controle de likes  
            this.posts[id+""] = 0;  
  
            socket.emit('makepost', postData);  
            socket.broadcast.emit('makepost', postData);  
        },  
  
        onMakeComment: function(socket, data) {  
            var id = Date.now(),  
                commentData = {  
                    author: socket.handshake.username,  
                    authorId: socket.id,  
                    timestamp: Date.now(),  
                    text: data.text,  
                    postId: data.postId,  
                    id: id  
                };  
  
            //cadastra o post no controle de likes  
            this.posts[id+""] = 0;  
  
            socket.emit('makecomment', commentData);  
            socket.broadcast.emit('makecomment', commentData);  
            console.log(commentData);  
        },  
  
        onLikePost: function(socket, data) {  
            var likeCount = this.posts[data.postId+""];  
  
            if(data.like) {  
                likeCount += 1;  
            } else {  
                likeCount -= 1;  
            }  
        }  
};
```

**Continuação:** Listagem 12. Regras de negócio da rede social no arquivo userhandling.js.

```
this.posts[data.postId+] = likeCount;

socket.emit('likepost', {postId: data.postId, numLikes: likeCount});
socket.broadcast.emit('likepost', {postId: data.postId, numLikes: likeCount});
};

onLikeComment: function(socket, data) {
  var likeCount = this.posts[data.commentId+'];

  if(data.like) {
    likeCount += 1;
  } else {
    likeCount -= 1;
  }

  this.posts[data.commentId+] = likeCount;

  socket.emit('likecomment', {commentId: data.commentId, numLikes: likeCount});
  socket.broadcast.emit('likecomment', {commentId: data.commentId, numLikes: likeCount});
};
```

Como havíamos comentado anteriormente, foram usados os mesmos nomes de eventos em ambos os lados do client e servidor, para garantir uma melhor legibilidade do código. Ou seja, se o server recebe um evento makepost, ao concluir a inclusão do post na timeline, ele informa o próprio usuário e os demais usuários com um evento de mesmo nome.

## Juntando as peças

Agora que escrevemos ambas aplicações no lado cliente e servidor, é hora de executá-la!

Vá para o diretório raiz rambook e execute o comando nodemon index.js.

Como mencionamos anteriormente, o Nodemon é uma ferramenta que executa um programa NodeJS, no nosso caso, index.js, e fica escutando por alterações no código. Toda vez que uma alteração é feita em um dos códigos contidos no diretório onde o Nodemon foi executado, ele recarrega toda a aplicação. Isso evita com que você tenha que parar o servidor e iniciá-lo novamente para cada alteração que fizer.

Após executar o Nodemon, você deverá ver uma tela semelhante à apresentada na Listagem 13.

**Listagem 13.** Servidor de websocket em ação.

```
chamb@selmy:~/Dropbox/projetos/tttserver$ nodemon index.js
20 Jan 21:14:06 - [nodemon] v0.7.10
20 Jan 21:14:06 - [nodemon] to restart at any time, enter `rs`
20 Jan 21:14:06 - [nodemon] watching: /home/chamb/Dropbox/projetos/tttserver
20 Jan 21:14:06 - [nodemon] starting `node index.js`
info - Socket.IO started
```

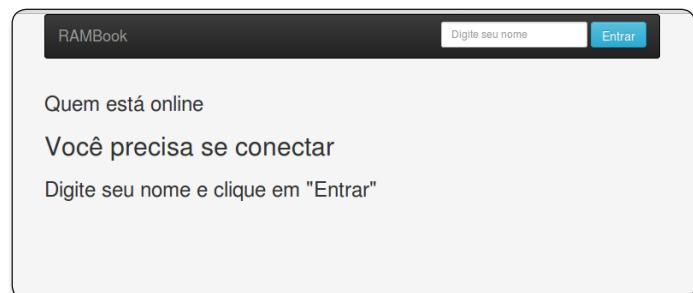
Agora podemos finalmente testar a aplicação, abrindo o browser e acessando a URL <http://localhost:9000/>.

Você deverá ver uma tela conforme exibido na Figura 2.

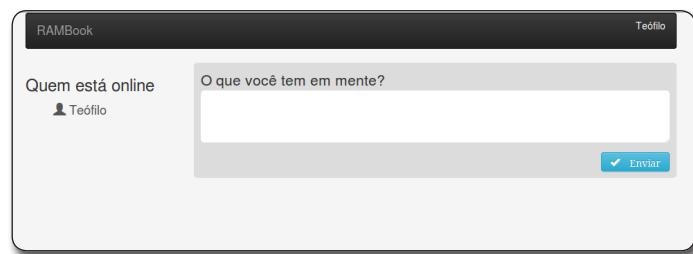
Perceba que o formulário de login leva um tempo para aparecer. Isso acontece pois inicialmente o elemento <div> que contém o formulário está oculto, e aguarda até que o client receba o evento on.(“connect”). Veja como ele é semelhante ao evento connection no lado do servidor.

Após aparecer o formulário de login, digite seu nome e clique em “Entrar”, e você verá uma tela idêntica à exibida na Figura 3.

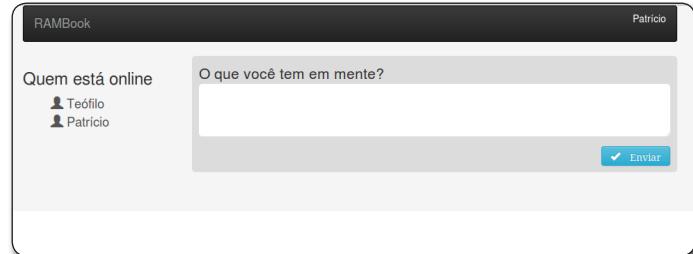
Se você abrir uma nova aba em seu browser e acessar a aplicação, poderá logar com um usuário de nome diferente e verá dois usuários conectados à mini rede social, conforme mostrado na Figura 4.



**Figura 2.** Tela de login da mini rede social



**Figura 3.** Tela da mini rede social após o login



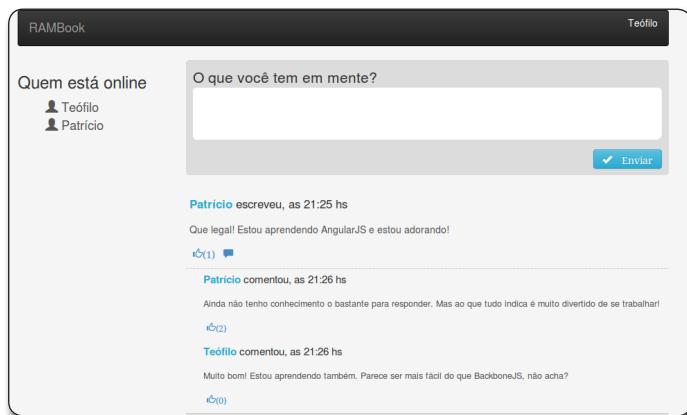
**Figura 4.** Usuários conectados simultaneamente na mini rede social

Agora, tente criar posts com um usuário, fazer comentários nesse post com outro usuário, curtir e “descurtir” posts e comentários. Você poderá notar que as informações estarão atualizadas em tempo real em ambas as abas do browser.

# NodeJS: Criando uma Rede Social

Faça testes com mais usuários, outros browsers, peça para seus amigos se conectarem em sua mini rede social e divirta-se! Você poderá ter um resultado tão bacana quanto o exemplo da **Figura 5**.

Esta é uma pequena amostra do poder do NodeJS e como ele, junto a APIs como o Socket.IO facilitam o desenvolvimento de aplicações em tempo real, graças a sua programação orientada a eventos.



**Figura 5.** Mostrando a aplicação em tempo real em ação

Nossa aplicação de exemplo é bastante divertida, porém, está longe de estar completa.

Tente implementar novas funcionalidades, como avisar ao usuário postador quando um de seus posts foram curtidos, ou validar os dados de forma a impedir o usuário de cadastrar posts em branco.

Se você já estiver se sentindo confortável com o NodeJS, tente criar a persistência dos dados de maneira a não perder os posts e comentários, e também criar um usuário e senha para impedir usuários de entrarem sem uma identidade bem definida. Quem sabe uma integração com a API do Facebook ou Twitter!

Em conclusão, pudemos ver uma amostra do que o NodeJS é capaz de fazer, e como ele tem sido largamente utilizado pela comunidade graças a ferramentas como o Grunt, Bower e Yeoman.

E, embora o NodeJS seja uma tecnologia muito empolgante, lembre-se de que ele não é uma bala de prata, e ainda que possa ser usado para desenvolver praticamente qualquer tipo de aplicação, cada tecnologia tem seu paradigma e especialidade, que, no caso do NodeJS é trabalhar com a programação orientada a eventos e concorrência.

## Autor



### Willian Carvalho

[o.chambs@gmail.com](mailto:o.chambs@gmail.com)

Programador desde 2000, trabalha com desenvolvimento para web, e já passou por ASP, PHP e principalmente Java. Formado em Tecnologia da Informação pela FASP. Atualmente é frontend tech lead na TOTVS onde trabalha com JavaScript e NodeJS.



## Links:

### Download do NodeJS

<http://nodejs.org/download/>

### Exemplos de uso do Twitter Bootstrap

<http://getbootstrap.com/getting-started/#examples>

### Site do Sublime Text

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

### Site do Socket.IO

<http://Socket.IO/>

### Site do RequireJS

<http://requirejs.org/>

### Site do NPM

<https://npmjs.org/>

# HTML5 Audio Tag: Crie um player de Áudio com HTML5

Entenda das tecnologias client-side para a criação de mecanismos usados em qualquer site

**N**a maior parte dos casos, executar músicas no plano de fundo de um site é uma má ideia, uma vez que o usuário não está preparado para lidar com o fator subjetividade. As músicas podem influenciar na decisão de se o usuário deve ou não continuar no site ou visitá-lo novamente no futuro, tal como o design de uma aplicação também se torna extremamente importante nesse ponto de vista. Mas, especificamente em algumas páginas da web, pode ser uma ideia bem interessante, tais como em sites relacionados ao tema música, blogs com podcasts e afins.

Por muito tempo, as formas mais comuns de se ter este tipo de conteúdo era utilizando um player de música em flash ou através de plug-ins específicos de programas instalados no computador do usuário, como o Windows Media Player, o Real Player ou o Quick Time, por exemplo. Até então, havia um grande problema quando da utilização dos plug-ins citados: a impossibilidade de customização do player, já que sua aparência iria seguir o padrão do software instalado no computador do usuário. Veja na **Figura 1** um exemplo de player de música com o plugin do Windows Media Player no Internet Explorer.

De tal forma, o trabalho de projeção do site ficaria comprometido, uma vez que o mesmo estaria preso a elementos gráficos e de design específicos que, certamente, não encaixariam com a ideia original do desenvolvedor para o design do site.

Algumas soluções alternativas poderiam ser usadas para burlar esse problema. A tecnologia Flash é um bom exemplo disso, já que não sofria com esse tipo de problema de customização, mas tinha os seus próprios. Desempenho era um deles.

## Fique por dentro

A criação de plugins customizáveis no mundo de desenvolvimento de softwares web sempre ocupou espaço de destaque entre os desenvolvedores front-end e web designers. Aliada ao uso das tecnologias JavaScript, HTML 5 (em sua nova versão com muito mais recursos), e bibliotecas de script prontas como o jQuery e afins, assim como às documentações bem elaboradas, a produtividade para criar e usar tais plugins cresce exponencialmente. Neste artigo será apresentado um tutorial de como utilizar tais tecnologias e plugins para criar um player de música para browser, explorando recursos famosos em aplicativos desktop que podem ser traduzidos para o âmbito web.



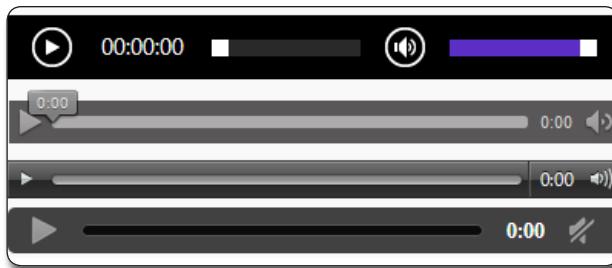
**Figura 1.** Plugin do Windows Media Player no Internet Explorer 10

O Flash nunca teve um desempenho muito bom fora do Windows e esse tipo de problema ultrapassa o universo dos computadores desktop chegando a acontecer também no mundo móvel. Além disso, o Flash não está presente em todos os dispositivos móveis. O iOS, por exemplo, nunca ofereceu suporte. Já o Android possui em algumas versões da plataforma, mas não é algo tão indicado para um smartphone, pelo alto consumo de recursos de CPU, bateria, etc.

Outra opção mais recente e interessante é a HTML5, que traz consigo a possibilidade de adicionar músicas e vídeos nas páginas sem a necessidade de plug-ins, deixando a responsabilidade de reproduzir os arquivos por conta do browser.

# HTML5 Audio Tag: Crie um player de Áudio com HTML5

A vantagem é que os esforços podem ser focados no desenvolvimento do restante do escopo, deixando todo o resto com o browser. Os pontos negativos são a incompatibilidade com alguns browsers, assim como o fato de que cada navegador assume um design próprio para exibir o player (**Figura 2**).



**Figura 2.** Players dos navegadores Internet Explorer, Firefox, Opera e Chrome (respectivamente)

Felizmente a HTML5 provê uma API bastante flexível que permite que a comunidade de desenvolvedores crie novas soluções para tornar o processo de inserção de conteúdo multimídia nas páginas mais simples e prático.

## A construção do player

O objetivo deste artigo será mostrar como fazer um player simples de áudio, utilizando uma biblioteca para a jQuery bastante flexível, chamada jPlayer, que funciona tanto para browsers modernos (com suporte à API de áudio do HTML5), quanto para os mais antigos. Essa compatibilidade de browsers é proporcionada através da utilização do Flash quando o browser do usuário não suporta a API de áudio do HTML5. Além disso, utilizando essa biblioteca, o player desenvolvido estará pronto para reproduzir áudio em dispositivos móveis.

O player que criaremos aqui foi testado nos navegadores Chrome, Opera, Internet Explorer 7+ e Firefox, assim como os browsers mobile padrão do Android 4, Opera mobile no Android 4, Internet Explorer no Windows Phone e Safari no iOS. O leitor pode ficar a vontade para testar em outros browsers de sua preferência.

Para conseguir entender e, consequentemente, aproveitar este tutorial ao máximo, você precisará ter alguns conhecimentos de JavaScript, tal como saber o que são e como usar objetos, arrays, funções etc., além de ter tido algum contato com a jQuery e trabalhar razoavelmente bem com HTML e CSS. Quanto mais personalização você quiser no seu player, mais você deverá saber sobre estas linguagens, especialmente JavaScript.

Conforme falado anteriormente, iremos utilizar essencialmente a biblioteca jQuery e o plugin jPlayer, os links de download para ambos podem ser encontrados na seção de **Links** no final desse artigo.

## Preparando o ambiente

Para facilitar o acompanhamento do tutorial, crie a seguinte estrutura de pastas para o projeto, conforme mostra a **Listagem 1**.

### Listagem 1. Estrutura de pastas para o projeto do player

```
01 |-- root  
02 |   |-- css  
03 |   |-- js  
04 |   |-- plugins  
05 |   |   |-- jplayer  
06 |   |-- songs
```

A estrutura é simples e bem intuitiva. Mas fique à vontade para modificá-la como quiser, caso não se sinta confortável com a distribuição ou julgue outra ser melhor.

Em seguida, selecione algumas músicas em formato mp3 de sua preferência e coloque-as na pasta “songs”. Para facilitar, deixe-as com nomes curtos, sem espaços ou acentos. Neste exemplo, foram utilizadas a “technologic.mp3” e a “human-after-all.mp3”.

## Marcação e Estilização

Primeiramente vamos criar um molde para marcar nossa página, que servirá como modelo para o local onde o player será inserido. Iremos utilizar um simples e que nos permita focar no objetivo principal, que é a criação do player, tal como exibido na **Figura 3**.



**Figura 3.** Modelo para marcar aparência do player

Vamos aos detalhes:

- **O item 1** contém os controles para o player. A partir deles conseguiremos iniciar e interromper a execução do som e percorrer a playlist.
- **O item 2** exibe a barra de progresso. A parte mais clara indica o quanto da música já foi reproduzido, e a parte escura quanto ainda está por vir.
- **O item 3** mostra ao usuário o nome da banda e o título da música que está sendo reproduzida.

Tanto os controles quanto todos os outros elementos podem ser estilizados usando CSS. Os elementos também podem ser removidos ou novos podem ser adicionados.

## Codificando

A codificação começa com o HTML. Crie um documento HTML na raiz do seu projeto e nomeie-o “index.html”. Adicione ao mesmo a marcação exibida na **Listagem 2**.

Esta é uma marcação bastante simples. A parte relevante para o player é a que está dentro da div que tem a classe “top-bar”.

Vamos analisar cada um dos elementos para entender seus papéis, identificando-os por suas classes CSS.

- **.player-controls**: este elemento abriga os controles do player. Não é obrigatório, mas ajuda a manter as coisas organizadas;
- **.player-prev, .player-pause, .player-stop, .player-next**: estes

elementos são, de fato, os controles do player. Para que o mesmo ofereça corretamente as funcionalidades esperadas, precisamos destes elementos com estas classes.

- **player:** este é um elemento (div) vazio dentro da página no player, isso porque o jPlayer precisa de um elemento do documento para criar o player e seus controles. Iremos então fornecer este, mas ele não terá nada de especial na nossa interface, por isso está vazio. É importante também fornecer um elemento específico, único e que os controles do player não fiquem dentro dele.

- **player-timeline, .player-timeline-control:** irão atuar como a barra de progresso, conforme o item 2 da **Figura 3**.

- **player-display, .layer-current-track:** iremos utilizar estes elementos para exibir informações da faixa atual ao usuário. Neste exemplo, serão exibidos apenas o nome do artista e o título da faixa.

#### Listagem 2. Marcação HTML inicial para a página do player

```
01 <!doctype html>
02 <html lang="pt-br">
03 <head>
04   <meta charset="utf-8" />
05   <title>jPlayer Tutorial</title>
06
07 </head>
08 <body>
09   <div class="top-bar">
10     <div class="container">
11       <div class="player-controls">
12         <span class="player-prev">Prev</span>
13         <span class="player-play">Play</span>
14         <span class="player-pause">Pause</span>
15         <span class="player-stop">Stop</span>
16         <span class="player-next">Next</span>
17     </div>
18   <div class="player"></div>
19   <div class="player-timeline">
20     <div class="player-timeline-control"></div>
21   </div>
22   <div class="player-display">
23     Playing: <span class="player-current-track"></span>
24   </div>
25 </div>
26 </div>
27 </body>
28 </html>
```

Após isso, é necessário agora incluir o arquivo de estilo que será usado na página. Para tanto, crie um novo arquivo na dentro da pasta “css” de nome “default.css” e inclua nele o código contigo na **Listagem 3**. Este código será responsável por adicionar as propriedades padrão para o corpo (body) da página, tais como margem e fonte.

Neste mesmo arquivo CSS, adicione agora o CSS para o restante dos componentes da página, de forma a deixar as coisas mais agradáveis visualmente. O código referente encontra-se na **Listagem 4**.

Finalmente, référencia no arquivo “index.html” a chamada ao arquivo CSS criado, adicionando o código contido na **Listagem 5** ao elemento <head> da mesma.

#### Nota

A inclusão foi feita em duas listagens para que o leitor possa atentar ao que é CSS principal e secundário. Na **Listagem 3** estamos estilizando o conteúdo geral da página, enquanto na **Listagem 4** estão sendo adicionados recursos de estilo para a barra superior e organização interna dos componentes.

#### Listagem 3. Conteúdo inicial do arquivo default.css

```
01 body {
02   padding: 0;
03   margin: 0;
04   font-family: sans-serif;
05 }
```

#### Listagem 4. Restante do conteúdo CSS do arquivo default.css

```
01 .container {
02   margin: auto;
03   width: 960px;
04 }
05 .top-bar{
06   width: 100%;
07   height: 30px;
08   background: #141414;
09   color: #888;
10   line-height: 30px;
11 }
12 .top-bar .player-controls {
13   float: left;
14 }
15 .top-bar .player-controls span{
16   cursor: pointer;
17   padding: 0 5px;
18   -webkit-transition: all linear 0.2s;
19 }
20 .top-bar .player-controls span:hover {
21   color: #EEE;
22 }
23 .top-bar .player-display {
24   float: left;
25   float: right;
26   margin-left: 50px;
27 }
28 .top-bar .player-display .player-current-track {
29   color: white;
30 }
31 .top-bar .player {
32   float: left;
33 }
34 .top-bar .player-timeline {
35   float: left;
36   max-width: 200px;
37   width: 200px;
38   height: 4px;
39   margin-top: 13px;
40   background: #555;
41   margin-left: 20px;
42 }
43 .top-bar .player-timeline-control {
44   height: 4px;
45   background: #999;
46 }
```

#### Listagem 5. Importação do arquivo default.css na página index.html

```
01 <link rel="stylesheet" href="css/default.css" />
```

# HTML5 Audio Tag: Crie um player de Áudio com HTML5

Não há nada de especial no nosso CSS, mas atente ao estilo das classes “.player-timeline” e “.player-timeline-control”. O elemento “player-timeline-control” irá ficar por cima do “player-timeline” e seu tamanho irá aumentar conforme a música for sendo reproduzida, enquanto o “player-timeline” irá exibir o tamanho total da reprodução. É interessante deixá-las com certo contraste de cores, para que o usuário consiga identificar o que está acontecendo.

Agora é hora de implementar o JavaScript. Paralelo a isso, precisaremos fazer uso também da jQuery. Neste tutorial utilizaremos a sua versão 1.8.3. Se houver versões mais recentes quando o leitor estiver lendo esse artigo, é possível que funcione também, assim como funciona com versões mais antigas. Além disso, utilizaremos o CDN (ver BOX 1) do Google para agilizar o processo de importação e uso da jQuery.

Veja na **Listagem 6** como importar a jQuery no nosso projeto, especificamente na tag <head> do arquivo HTML.

## BOX 1.CDN

Significa Content Delivery Network ou Rede de Distribuição de Conteúdo. É um sistema distribuído entre múltiplos servidores para fornecer aos utilizadores acesso rápido e confiável a determinado conteúdo. Carregando a jQuery pelo CDN da Google torna, em vários casos, desnecessário que o browser do cliente faça o download do script, já que ele pode ter acessado algum outro site que o utiliza também e armazenado em cache.

## Listagem 6. Importando a jQuery na página

```
01 <script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js">
</script>
```

Com a jQuery pronta, é hora de baixar os arquivos da biblioteca jPlayer. Efetue o download dos arquivos citados através do link disponibilizado na seção **Links** e coloque-os no diretório “js/plugins/jplayer”. O primeiro arquivo baixado corresponde à “biblioteca jPlayer”. O segundo, o “swf”, é o responsável por reproduzir os arquivos de áudio quando o browser não oferecer o suporte à API de áudio do HTML5.

Adicione também a chamada ao import do arquivo JavaScript da biblioteca jPlayer, tal como na **Listagem 7**.

Para que o player funcione, todavia, é necessário que se faça uso de um script JavaScript personalizado, utilizando as referências às funções disponibilizadas pela biblioteca jPlayer. Para tal, crie um arquivo JavaScript chamado “default.js” no diretório “js” da estrutura de pastas e acrescente o código da **Listagem 8** ao mesmo. Adicione também a chamada import na página index.html, tal como na **Listagem 9**.

A Playlist nada mais é do que um vetor de objetos que representam as músicas. Cada objeto deve representar um arquivo de áudio que deverá ser reproduzido. No código da **Listagem 8** podemos perceber a criação de objetos JavaScript representando cada uma das músicas a serem exibidas, com seus atributos (artist, title, etc) no formato “chave-valor”. A informação mais importante é a que está na chave “mp3”, que representa a url do caminho onde o arquivo de som no formato MP3. O jPlayer consegue lidar com vários

tipos de arquivos de som e vídeo, como mp3, m4a, m4v e oga. Se você pretende utilizar o player dentro de um CMS (ver **BOX 2**), por exemplo, é bem mais provável que seus clientes tenham os arquivos em mp3 do que em outros formatos, por essa e muitas outras razões este foi o formato escolhido. A documentação do plugin fornece mais detalhes sobre como utilizar vários formatos.

Repare que na chave mp3, colocamos os nomes dos arquivos adicionados à pasta “songs” no início do artigo. Se seus arquivos de música são diferentes (e certamente serão), então modifique os valores das chaves para contemplar os mesmos. As demais informações definidas nos nossos objetos de música são completamente opcionais, mas é interessante mantê-las, pois é daí que extrairemos a informação que será exibida ao usuário na interface da página. Você pode adicionar, remover ou alterar estas outras chaves como quiser, tudo depende do que você quer exibir para o usuário.

## BOX 2.CMS

Significa Content Management System ou Sistema de Gerenciamento de Conteúdo. É um sistema que permite a adição e edição de informações numa interface centralizada. O Wordpress é um exemplo que, após a instalação, permite que os usuários alterem o conteúdo à vontade, mesmo sem entender de programação.

## Listagem 7. Importando o arquivo JavaScript do jPlayer

```
01 <script type="text/JavaScript" src="js/plugins/jplayer/jquery.jplayer.min.js">
</script>
```

## Listagem 8. Script personalizado com configurações da playlist

```
01 $(function() {
02   // Definir playlist
03   var playlist = [
04     artist:'Daft Punk',
05     title:'Technologic',
06     mp3:'songs/technologic.mp3'
07   ],{
08     artist:'Daft Punk',
09     title:'Human After All',
10     mp3:'songs/human-after-all.mp3'
11   }];
12});
```

## Listagem 9. Importando arquivo default.js à página index.html

```
01.<script type="text/JavaScript" src="js/default.js"></script>
```

Agora criaremos duas variáveis que irão nos ajudar a controlar a reprodução da playlist. A primeira é a **currentTrack**, dê a ela o valor 0. Ela irá controlar qual é a entrada na playlist que está em execução no momento. O zero foi definido como valor padrão, que irá representar a primeira música no nosso array “playlist”. Se você quiser que a reprodução comece pela segunda, terceira ou outra posição qualquer, basta alterar este número, embora seja recomendado que você organize sua playlist exatamente como pretende que as músicas sejam reproduzidas, para facilitar o entendimento do código posteriormente.

A segunda variável será criada com o objetivo de obter, de forma prática e rápida, o número de faixas presentes na playlist. A **Listagem 10** exemplifica como fazer isso. Note que a inclusão destes códigos deve ser feita logo abaixo da criação do array “playlist” da **Listagem 9**.

Logo abaixo, inclua o código referente à **Listagem 11**, que será responsável por criar o player, propriamente dito.

#### **Listagem 10.** Variáveis auxiliares para controlar faixa atual e número de faixas

```
01 var currentTrack = 0;
02 var numTracks = playlist.length;
```

#### **Listagem 11.** Código de criação do player

```
01 var player = $("player").jPlayer({
02   ready: function () {
03     // configura a faixa inicial do jPlayer
04     player.jPlayer("setMedia", playlist[currentTrack])
05
06     // reproduzir a faixa atual. Se não quiser que o player comece a tocar
07     // automaticamente
08     // retirar esta linha
09     player.playCurrent();
10   },
11   ended: function() {
12     // quando terminar de tocar uma música, ir para a próxima
13     $(this).playNext();
14   },
15   play: function(){
16     // quando começar a tocar, escrever o nome da faixa sendo executada
17     $('.player-current-track').text(playlist[currentTrack].artist+' - '+playlist
18     [currentTrack].title);
19   },
20   swfPath: 'js/plugins/jplayer/',
21   supplied: "mp3",
22   cssSelectorAncestor: "",
23   cssSelector: {
24     play: ".player-play",
25     pause: ".player-pause",
26     stop: ".player-stop",
27     seekBar: ".player-timeline",
28     playBar: ".player-timeline-control"
29   },
30   size: {
31     width: "1px",
32     height: "1px"
33   }
34});
```

Resumindo, foi criada uma variável (player) para abrigar o nosso player e utilizamos a função **jPlayer()** na div que criamos para receber os elementos responsáveis pela execução do arquivo de áudio. O jPlayer cria outros elementos dentro da mesma que possibilitam a reprodução do som, mas não precisamos nos preocupar com eles no momento. O parâmetro que passamos para a função **jPlayer()** é um objeto com a configuração necessária para o funcionamento do player. Vamos analisar cada uma dessas opções:

- **ready:** Evento disparado quando o player é construído e está pronto para receber instruções. Neste caso, queremos o player com “autoplay”, então neste momento definimos a faixa atual e chamamos o método do nosso player responsável pela reprodução da faixa corrente. Este método é o “**playCurrent()**”. Ele ainda não

foi definido, mas chegaremos nele logo mais. Se não quiser que a faixa comece a ser reproduzida automaticamente, basta remover a chamada ao método **playCurrent()**.

- **ended:** Este evento é disparado sempre que a reprodução de alguma faixa chega ao fim. Utilizamos para que a próxima faixa da playlist seja definida e a reprodução se inicie. Para isso, chamamos o método “**playNext()**”, que irá identificar a próxima faixa e executá-la.

- **play:** Este evento é disparado no início da reprodução de um arquivo de áudio. Este é o momento ideal para atualizarmos o elemento na nossa página, que irá exibir o nome do artista e o título da faixa. O que fazemos aqui é simplesmente buscar esta informação da nossa playlist e escrever no elemento “**.player-current-track**”, que definimos na nossa marcação. Nesse caso, como já falado, apenas o nome do artista e o título da faixa serão exibidos, mas você pode adicionar qualquer outra informação.

- **swfPath:** O jPlayer utiliza o elemento **<audio>** do HTML5 para reproduzir as músicas. Porém, em browsers que não tem suporte, ele utiliza um pequeno player em Flash. Ao executar o download do jPlayer, junto dele nota-se a existência de um arquivo “**jplayer.swf**”. Você precisa salvá-lo em algum lugar dentro do seu projeto e colocar o endereço do diretório nessa propriedade. No nosso caso, colocamos o swf na mesma pasta da biblioteca (**/js/plugins/jplayer**), então colocamos o caminho da pasta do jPlayer relativo ao documento no qual estamos inserindo o mesmo.

- **supplied:** A propriedade “**supplied**” especifica os tipos de arquivos que iremos utilizar. Como os arquivos definidos serão apenas mp3, então o conteúdo da mesma propriedade será uma string contendo ‘mp3’. Você pode especificar mais de um tipo. Por exemplo, se tiver arquivos mp3 e m4a, pode usar o valor ‘**mp3,m4a**’.

- **cssSelectorAncestor:** Esta propriedade define o elemento base onde os controles serão colocados quando o player for criado. Como estamos utilizando uma marcação completamente personalizada e flexível, iremos definir uma string vazia para termos controle sobre os elementos, assim o jPlayer não irá criar nada na nossa marcação.

- **cssSelector:** A propriedade “**cssSelector**” permite que especifiquemos o que cada controle pode fazer. Precisaremos definir um objeto onde as chaves representam a ação e os valores representam o seletor do elemento responsável. As seguintes chaves serão utilizadas:

- **play:** passando o elemento **.player-play**, que é o botão play;
- **pause:** passando o elemento **.player-pause**, que é o botão pause;
- **stop:** passando o elemento **.player-stop**, que é o botão stop;
- **seekBar:** passando o elemento **.player-timeline**, que representa a barra de progresso. Esta é a que mostra o tempo total do som;
- **playBar:** passando o elemento **.player-timeline-control**, que mostra o progresso da execução. O usuário poderá ir para momentos específicos da faixa clicando sobre este controle;

# HTML5 Audio Tag: Crie um player de Áudio com HTML5

- size: A propriedade “size” define o tamanho do player criado pelo jPlayer. Dentro deste elemento, o jPlayer irá criar o elemento <audio> ou incluir o Jplayer.swf.

Nas explanações anteriores sobre cada um dos atributos do player, você pode notar a referência a três métodos para controle da navegação entre as músicas. O primeiro deles é o “playCurrent()”. Este método será responsável por solicitar ao jPlayer a reprodução da música atual. Para implementá-lo, vamos utilizar o método jPlayer() duas vezes. Na primeira, serão passados dois parâmetros: a string “setMedia” para dizer ao jPlayer qual arquivo será trabalhado a partir de então, e o segundo, o objeto que representa o arquivo (neste caso o item “currentTrack” dentro da playlist). Na segunda chamada ao método jPlayer(), apenas o parâmetro “play” será passado por parâmetro, este que informa ao jPlayer que ele já pode reproduzir o arquivo que está “setado”.

## Nota

Todos os métodos de navegação serão criados adicionando-se uma nova function às propriedades playX do objeto player.

O segundo método é o “playNext()” que, por sua vez, irá identificar qual é a próxima faixa, defini-la como faixa atual e utilizar o método que criamos no passo anterior (playCurrent) para executá-la. A variável numTracks, criada na **Listagem 10**, vai auxiliar a reiniciar a execução da playlist quando ela chegar ao fim. Além disso, é importante ater-se ao estado atual de execução, considerando que se a música atual for a última da playlist, então iremos executar a primeira, senão executaremos a próxima.

Finalmente, o terceiro método, o “playPrevious()”, será responsável por voltar a execução para a faixa anterior. Se a faixa atual for a primeira da nossa playlist, então ele irá executar a última.

A **Listagem 12** exemplifica a aplicação desses três métodos. Adicione-a ao final da **Listagem 11**.

Por fim iremos tratar os cliques nos botões de “próxima faixa” e “anterior”. Para isso, será necessário ouvir o evento “click” nesses elementos e chamar a função apropriada à ação desejada pelo usuário, utilizando as funções definidas anteriormente para a navegação. Veja no código da **Listagem 13** como fazer isso.

Agora é testar no seu navegador. Se tudo estiver correto, você irá ouvir sua música sendo reproduzida e poderá avançar e voltar na sua playlist. O resultado pode ser visualizado na **Figura 3**. Caso a música não esteja sendo reproduzida, verifique se seguiu todos os passos corretamente. Verifique também o console do seu navegador. Ele é muito útil para identificar problemas no código em execução e o jPlayer costuma colocar várias informações lá quando encontra algum erro.

Depois que o desenvolvedor se acostuma com a biblioteca, fazer alterações no player se tornam bem simples. Por exemplo, se quiser adicionar o nome do álbum ao playlist, modifique o array playlist para que ele fique tal como na **Listagem 14**.

Agora nossa playlist contém o título do álbum de cada faixa. E para que o player funcione, é preciso modificar o que fazemos no evento play.

Localize a linha 15 da **Listagem 11** e substitua-a pelo código da **Listagem 15**.

## Listagem 12. Implementação dos métodos playCurrent(), playNext() e playPrevious()

```
01 player.playCurrent = function() {  
02   player.jPlayer("setMedia", playlist[currentTrack]).jPlayer("play");  
03 }  
04  
05 player.playNext = function() {  
06   currentTrack = (currentTrack == (numTracks - 1)) ? 0 : ++currentTrack;  
07   player.playCurrent();  
08 };  
09  
10 player.playPrevious = function() {  
11   currentTrack = (currentTrack == 0) ? numTracks - 1 : --currentTrack;  
12   player.playCurrent();  
13 };
```

## Listagem 13. Ouvintes de cliques nos botões de próxima e anterior

```
01 $(".player-next").click(function() {  
02   player.playNext();  
03 });  
04  
05 $(".player-prev").click(function() {  
06   player.playPrevious();  
07 });
```

## Listagem 14. Playlist com inclusão do nome do álbum

```
01 var playlist = [{  
02   artist: 'Daft Punk',  
03   title: 'Technologic',  
04   album: 'Human After All',  
05   mp3: 'songs/technologic.mp3'  
06 }, {  
07   artist: 'Daft Punk',  
08   title: 'Human After All',  
09   album: 'Human After All',  
10   mp3: 'songs/human-after-all.mp3'  
11 }];
```

## Listagem 15. Código de alteração para exibição do nome do álbum

```
01 $(".player-current-track").text(playlist[currentTrack].artist + '-'  
  + playlist[currentTrack].title + '-' + playlist[currentTrack].album);
```

Pronto! Agora o player irá mostrar também o nome do álbum.

A mesma coisa pode ser feita se você quiser colocar a capa do álbum. Basta colocar a url da capa do álbum na playlist e a imagem no lugar em que você queira que apareça. Pode criar dinamicamente um novo elemento img ou pode reutilizar um criado especificamente pra isso. Tudo depende de como você queira montar o seu player.

## Criando uma playlist

Com posse de todos estes conhecimentos você já será capaz de manipular diversos recursos interessantes do plugin jPlayer.

Uma reclamação muito comum entre desenvolvedores das mais diversas linguagens está relacionada à quantidade de código necessário para se desenvolver determinadas soluções, muitas delas simplistas e comuns. A melhor forma de mensurar isso é experiência. Quanto mais experiência você tiver em determinada tecnologia, mais produtivo será com relação ao tempo que leva pra fazer as coisas simplistas e/ou mais complexas. Por exemplo, considere o cenário de que não ficou satisfeito apenas com a criação de um player básico. Suponha que você necessita de algo um pouco mais arrojado, uma funcionalidade onde você possa criar uma lista de reprodução e disponibilizar para o usuário a visão de que ele pode escolher qualquer música dentre as opções disponíveis, suponha que você queira criar uma playlist.

As características de uma playlist não mudam muito do player que criamos até então, uma vez que a diferença mais significativa é a exibição da lista de músicas junto aos botões de execução, tal como pode ser visto em players como o Windows Media Player para Windows.

#### Nota

É preciso saber a distinção entre player e playlist. Um player é a ferramenta que executa as músicas. A playlist é apenas uma lista de execução.

O **JPlayerPlaylist** é um add-on (extensão) para o plugin jPlayer. Ele foi feito com o objetivo de complementar o plugin do jPlayer no que concerne ao desenvolvimento especificamente de playlists. Trabalhar com ele é tão simples quanto usar o jPlayer, os comandos são semelhantes, a forma de implementar a estrutura de chamadas também, entretanto ele exigirá que você tenha uma estrutura HTML montada de acordo com as definições dos arquivos JavaScript e CSS do mesmo.

O primeiro passo é efetuar o download do plugin, um arquivo de extensão .js. Você poderá encontrar o link para download na seção **Links** no final do artigo. Coloque o arquivo baixado na pasta “js” do projeto que estamos desenvolvendo.

Após isso, crie um novo arquivo HTML na raiz do projeto e nomeie-o como “playlist.html”. O arquivo conterá o conteúdo de uma nova página HTML preparada para executar um player mais completo. Com lista de músicas, links para cada uma, botões de próximo, anterior, pausa e play, e até opções de aumentar e diminuir o volume da música tocada.

Veja na **Listagem 16** o conteúdo da página.

Note que o código está comentado, o suficiente para que você possa embasar o que cada parte faz e as responsabilidades de cada uma.

Note também que adicionamos mais músicas à lista disponível. Se for utilizar suas próprias músicas, lembre-se de mudar corretamente os nomes dos arquivos no JavaScript do início da página, assim como as descrições nos links ao final do artigo.

Após isso, você deve adicionar os arquivos de estilo (CSS) e a imagem nas pastas correspondentes.

Os mesmos arquivos podem ser encontrados no link de download deste mesmo artigo. Certifique-se de criar também uma pasta “img” dentro da raiz do projeto para guardar todas as imagens usadas.

Se tudo tiver sido feito corretamente, execute a página HTML e você verá como resultado o player exibido na **Figura 4**.



**Figura 4.** Resultado da criação da playlist com a extensão jPlayerPlaylist

## Mais sobre o jPlayer

### Formatos de áudio

Os formatos de mídia que são essenciais para o JPlayer são aqueles que são suportados por ambas soluções Flash e os navegadores HTML5 que não suportam Flash, tais como iOS. É importante que um desses formatos seja fornecido para JPlayer para que os navegadores populares sejam capazes de reproduzir as mídias. Depois de um dos formatos essenciais ter sido fornecido, formatos de contrapartida adicionais podem ser fornecidos para aumentar o apoio cross-browser da solução HTML5. A opção jPlayer ({"supplied": "formats"}) oferece mais detalhes de como esse processo funciona. Em suma, pelo menos um dos formatos de áudio e vídeo deve ser fornecido pelo browser que está a executar o player, são eles: MP3 ou M4A para áudio e MP4 para vídeo.

### Regras de segurança Flash

As regras de segurança para o arquivo SWF do jPlayer foram melhoradas usando o código da **Listagem 17** que pode ser chamado a partir de qualquer domínio.

Geralmente, você irá fazer o upload do arquivo SWF com o arquivo JavaScript para um diretório chamado “js” no seu domínio. Use a opção construtor jPlayer( {"swfPath" : caminho }) para alterar o caminho.

Tentar executar o jPlayer localmente em seu computador irá gerar violações de segurança em Flash e você precisa permitir o acesso de arquivos locais usando o Gerenciador de configurações do Flash. Consulte a Ajuda do Flash Player para obter mais informações.

Para desenvolver localmente, instale um servidor no seu sistema, como o Apache, para permitir um host local no seu computador.

# HTML5 Audio Tag: Crie um player de Áudio com HTML5

Listagem 16. Página HTML da playlist

```
01 <!doctype html>
02 <html lang="pt-br">
03 <head>
04   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
05   <title>JPlayer Playlist - Web Magazine Devmedia</title>
06   <link href="css/jplayer.blue.monday.css" rel="stylesheet" type="text/css">
07   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js">
08     </script>
09   <script type="text/JavaScript" src="http://demo.chapmanit.com/jplayerPlaylist/
10     js/jquery.jplayer.min.js"></script>
11   <script type="text/JavaScript">
12   // Variável responsável por guardar o item atual de execução
13   var playItem = 0;
14
15  /*
16    Lista com todas as músicas a serem executadas na playlist.
17    Em uma aplicação dinâmica, os valores provavelmente serão montados a
18    partir de uma linguagem server side.
19 */
20  var minhaPlaylist = [
21    {name:"Daft Punk - Human After All", mp3:"songs/human-after-all.mp3"},
22    {name:"Amy Winehouse - You Know I'm No Good",
23      mp3:"songs/you-know-im-no-good.mp3"},
24    {name:"Black Eyed Peas - Shut Up", mp3:"songs/shut-up.mp3"},
25    {name:"Nightwish - Ghost River", mp3:"songs/ghost-river.mp3"},
26    {name:"Daft Punk - Technologic", mp3:"songs/technologic.mp3"}
27 ];
28
29 /* Cópias locais para os seletores jQuery, apenas para performance */
30 // Guarda o tempo atual de execução
31 var jpTempoExecucao = $("#jplayer_tempo_execucao");
32 // Guarda o tempo total de execução
33 var jpTempoTotal = $("#jplayer_tempo_total");
34
35 // Função de criação e configuração do player.
36 $("#jquery_jplayer").jPlayer({
37   ready: function() {
38     exibirPlaylist();
39     playListInit(true); // Parâmetro é um para autoplay.
40   },
41   oggSupport: false
42 }
43 // Configurações gerais do player
44 .jPlayer("onProgressChange", function(loadPercent, playedPercentRelative,
45   playedPercentAbsolute, playedTime, totalTime) {
46   jpTempoExecucao.text($.jPlayer.convertTime(playedTime));
47   jpTempoTotal.text($.jPlayer.convertTime(totalTime));
48 }
49
50 // Captura o evento de clique para o botão de anterior
51 $("#jplayer_anterior").click(function() {
52   playListAnterior();
53   $(this).blur();
54   return false;
55 });
56
57 // Captura o evento de clique para o botão de próximo
58 $("#jplayer_proximo").click(function() {
59   playListProximo();
60   $(this).blur();
61   return false;
62 });
63
64 // Método interno de montagem e exibição da playlist
65 function exibirPlaylist() {
66   $("#jplayer_playlist ul").empty();
67   for (i=0; i < minhaPlaylist.length; i++) {
68     var listItem = (i == minhaPlaylist.length-1) ? "
69       <li class='jplayer_playlist_ultimo_item'>": "<li>";
70     listItem += "<a href='"+$("#jplayer_playlist_item_"+i+"").attr("href")+
71       + minhaPlaylist[i].name +"'>" + minhaPlaylist[i].name + "</a> (<a id='jplayer_playlist_get_mp3_"+i+"'
72       href='"+minhaPlaylist[i].mp3+"'" + "tabindex='1'" + "mp3</a>)" + "</li>";
73     $("#jplayer_playlist ul").append(listItem);
74     $("#jplayer_playlist_item_"+i).data("index", i).click(function() {
75       var index = $(this).data("index");
76       if (playItem != index) {
77         mudarPlaylist(index);
78       } else {
79         $("#jquery_jplayer").jPlayer("play");
80       }
81     });
82     var index = $(this).data("index");
83     $("#jplayer_playlist_item_"+index).trigger("click");
84     $(this).blur();
85     return false;
86   });
87 }
88
89 // Inicializa a playlist
90 function playListInit(autoplay) {
91   if(autoplay) {
92     mudarPlaylist(playItem);
93   } else {
94     playListConfig(playItem);
95   }
96 }
97
98
99 // Configura a playlist (quando a mesma não está por padrão como autoplay)
100 function playListConfig(index) {
101   $("#jplayer_playlist_item_"+index).removeClass("jplayer_playlist_current");
102   parent().removeClass("jplayer_playlist_current");
103   $("#jplayer_playlist_item_"+index).addClass("jplayer_playlist_current");
104   parent().addClass("jplayer_playlist_current");
105   playItem = index;
106   $("#jquery_jplayer").jPlayer("setFile", minhaPlaylist[playItem].mp3,
107   minhaPlaylist[playItem].ogg);
108
109   function mudarPlaylist(index) {
110     playListConfig(index);
111     $("#jquery_jplayer").jPlayer("play");
112   }
113
114   // Executa a próxima faixa
115   function playListProximo() {
116     var index = (playItem+1 < minhaPlaylist.length) ? playItem + 1 : 0;
117     mudarPlaylist(index);
118   }
119
120   // Executa a faixa anterior
121   function playListAnterior() {
122     var index = (playItem-1 >= 0) ? playItem-1 : minhaPlaylist.length-1;
123     mudarPlaylist(index);
124   }
125 }
```

## Continuação: Listagem 16. Página HTML da playlist

```
126 </head>
127 <body>
128 <!-- Código para forçar a execução da primeira música quando a página abre. -->
129 <div id="jquery_jplayer" style="position: absolute; top: 0px; left: -9999px;">
130   <audio id="jqjp_audio_0" preload="none" src="songs/human-after-all.mp3">
131     </audio>
132   <div id="jqjp_force_0" style="text-indent: -9999px;">0.3245763930026442
133   </div>
134 
135 <div class="jp-playlist-player">
136   <div class="jp-interface">
137     <ul class="jp-controls">
138       <li><a href="#" id="jplayer_play" class="jp-play" tabindex="1" title="Executar">play</a></li>
139       <li><a href="#" id="jplayer_pause" class="jp-pause" tabindex="1" style="display: block;" title="Pausar">pause</a></li>
140       <li><a href="#" id="jplayer_stop" class="jp-stop" tabindex="1" title="Parar">stop</a></li>
141       <li><a href="#" id="jplayer_volume_min" class="jp-volume-min" tabindex="1" title="Mínimo">min volume</a></li>
142       <li><a href="#" id="jplayer_volume_max" class="jp-volume-max" tabindex="1" title="Máximo">max volume</a></li>
143       <li><a href="#" id="jplayer_anterior" class="jp-previous" tabindex="1" title="Anterior">previous</a></li>
144       <li><a href="#" id="jplayer_proximo" class="jp-next" tabindex="1" title="Próximo">next</a></li>
145     </ul>
146   <div class="jp-progress">
147     <div id="jplayer_load_bar" class="jp-load-bar" style="width: 0%;">
148       <div id="jplayer_play_bar" class="jp-play-bar" style="width: 0%;">
149     </div>
150   </div>
```

```
151   <div id="jplayer_volume_bar" class="jp-volume-bar">
152     <div id="jplayer_volume_bar_value" class="jp-volume-bar-value" style="width: 80%;"></div>
153   </div>
154   <div id="jplayer_tempo_execucao" class="jp-play-time">00:00</div>
155   <div id="jplayer_tempo_total" class="jp-total-time">00:00</div>
156 </div>
157   <div id="jplayer_playlist" class="jp-playlist">
158     <ul>
159       <li class="jplayer_playlist_current">
160         <a href="#" id="jplayer_playlist_item_0" tabindex="1" class="jplayer_playlist_current">Daft Punk - Human After All</a>
161       </li>
162       <li>
163         <a href="#" id="jplayer_playlist_item_1" tabindex="1">Amy Winehouse - You Know I'm No Good</a>
164       </li>
165       <li>
166         <a href="#" id="jplayer_playlist_item_2" tabindex="1">Black Eyed Peas - Shut Up</a>
167       </li>
168       <li>
169         <a href="#" id="jplayer_playlist_item_3" tabindex="1">Nightwish - Ghost River</a>
170       </li>
171       <li class="jplayer_playlist_ultimo_item">
172         <a href="#" id="jplayer_playlist_item_5" tabindex="1">Daft Punk - Techonologic</a>
173       </li>
174     </ul>
175   </div>
176 </div>
177 </body>
178</html>
```

## Listagem 17. Código de segurança Flash

```
01 flash.system.Security.allowDomain (" * ");
02 flash.system.Security.allowInsecureDomain (" * ");
```

## Tipos MIME

Quando a sua aplicação lida com a publicação final, ou seja, a publicação em um servidor você deve ter sempre em mente que o ambiente de publicação final é, em muitos casos, diferente do ambiente de desenvolvimento. Em vista disso, é necessário que o seu servidor tenha todas as configurações corretamente feitas para evitar conflitos entre os tipos manipulados.

MIME (*Multipurpose Internet Mail Extensions*) é um padrão de internet que é utilizado para descrever o conteúdo de vários arquivos. Enquanto o nome MIME quer dizer "Mail", ele também é usado para páginas da web.

O tipo MIME para HTML é "text/html", para arquivos Excel é "application/vnd.ms-excel" e etc.

Tipos MIME são definidos no HTML pelo atributo "type" em links, objetos e tags de script e estilo.

O servidor do seu domínio deve dar o tipo MIME correto para todas as URLs de mídia. A falta o tipo MIME correto vai fazer

com que a mídia não funcione corretamente em alguns navegadores HTML5. Esta é uma causa comum de problemas que afetam somente o Firefox e o Opera. Outros navegadores são menos rigorosos, mas o tipo MIME sempre deve ser verificado se está correto, dado se você estiver tendo problemas para ter a mídia executando em qualquer navegador.

## Listagem 18. Exemplos de Tipos MIME comuns para o plugin jPlayer

```
MP3: audio/mpeg
MP4: audio/mp4 video/mp4
OGG: audio/ogg video/ogg
WebM: audio/webm video/webm
WAV: audio/wav
```

Veja na Listagem 18 alguns exemplos comuns para o jPlayer. Se você usar uma extensão comum para mídias de áudio e vídeo, como audio.mp4 e video.mp4 por exemplo, então simplesmente use a versão em vídeo do tipo MIME para ambos, isto é, "video/mp4".

Em servidores Apache, você pode usar o arquivo de extensão ".htaccess" para definir o tipo MIME com base na extensão do arquivo (Listagem 19).

# HTML5 Audio Tag: Crie um player de Áudio com HTML5

## Listagem 19. Tipos MIME para configuração do Apache Server

```
#AddType TIPO/EXTENSÃO DO SUBTIPO
```

```
AddType audio/mpeg mp3  
AddType audio/mp4 m4a  
AddType audio/ogg ogg  
AddType audio/ogg oga  
AddType audio/webm webma  
AddType audio/wav wav
```

```
AddType video/mp4 mp4  
AddType video/mp4 m4v  
AddType video/ogg ogv  
AddType video/webm webm  
AddType video/webm webmv
```

## Não use GZIP nas Mídias

Desative a codificação GZIP (ver **BOX 3**) de todos os arquivos de mídia. Os arquivos de mídia já estão compactados e o GZIP só vai perder CPU em seu servidor. O plugin do Adobe Flash vai ter problemas se você usar GZIP nas mídias. Não GZIP o arquivo Jplayer.swf também. Sinta-se livre para GZIP o JavaScript.

### BOX 3. GZIP

GZIP é um software criado com a finalidade de comprimir dados sem perdê-los. Baseado no algoritmo DEFLATE, ele gera um arquivo comprimido de extensão ".gz". O nome se origina da abreviação dos nomes GNU zip e é comumente usado em sistemas UNIX.

## Requisições Byte-Range

Seu servidor deve habilitar solicitações de "extensão". Isso é fácil de verificar vendo se a resposta do seu servidor inclui os "Accept-Ranges" em seu cabeçalho. A maioria dos navegadores HTML5 permitirá a busca das novas posições dos arquivos durante um download, por isso, o servidor deve permitir que o novo Range a seja solicitado.

A não aceitação dos pedidos de intervalo de bytes vai causar problemas em alguns navegadores HTML5. Muitas vezes a duração não pode ser lida a partir do arquivo, tal como alguns formatos requerem que o início e o fim do arquivo sejam lidos para saber a sua duração. O Chrome tende a ser o navegador que tem a maioria dos problemas se a requisição de Range não estiver habilitada no servidor, mas todos os navegadores terão algum problema. Mesmo que seja só isso, você tem que esperar por todas as mídias antes de ir até o fim com as indagações.

Este problema é conhecido por afetar os servidores Jetty 6 com a sua configuração padrão. Uma função PHP foi escrita pela comunidade jPlayer que pode servir arquivos de mídia com suporte a solicitações de extensões.

## Protegendo suas mídias

Tenha cuidado ao tentar restringir o acesso aos seus arquivos de mídia. A URL da mídia deve ser acessível através da internet pelo usuário e sua resposta deve estar no formato esperado.

Usar a resposta do servidor para desativar o cache local de mídia pode causar problemas com alguns navegadores HTML5. Isso pode fazer com que a duração das mídias seja desconhecida, o que irá mostrar um "NaN" (Not a Number) quando solicitar o tempo de duração do jPlayer.

Há inúmeras formas de se colocar áudio em páginas na web. Aqui exploramos uma forma simples e flexível, que possui um mecanismo com detalhes visuais personalizáveis e lista de arquivos para reprodução.

Os exemplos no início do artigo mostram como a biblioteca jPlayer pode ser utilizada de várias formas, desde tocar trechos de músicas em lojas online, quanto em aplicações web das mais diversas finalidades.

## Autor



Luis Henrique

[luish.faria@msn.com](mailto:luish.faria@msn.com) - <http://www.luque.cc/>

Formado em Licenciatura em Computação, atua com desenvolvimento para a web desde 2009. Atualmente é Analista de Sistemas, trabalhando principalmente com PHP, MySQL e também freelancer, atuando com HTML, CSS e JavaScript.



## Links:

### Documentação oficial do jPlayer.

[www.jplayer.org/latest/developer-guide/](http://www.jplayer.org/latest/developer-guide/)

### Site da jQuery.

[www.jquery.com](http://www.jquery.com)

### Site do jPlayer.

[www.jplayer.org](http://www.jplayer.org)

### Página do jPlayerPlaylist.

[www.jplayer.org/latest/demo-02-jPlayerPlaylist/](http://www.jplayer.org/latest/demo-02-jPlayerPlaylist/)

# Programando em HTML5

## Aprofunde-se nas novidades que vão muito além de meras novas tags

**A**o ser anunciada a versão 5 do padrão HTML, inicialmente parte do mercado não recebeu com grande entusiasmo, acostumado a receber poucos recursos novos de uma versão para outra. A verdade é que, desde que a versão 4.0 foi lançada em 1997, poucos avanços aconteceram nos dez anos seguintes. O padrão foi atualizado para 4.01 (praticamente uma errata) e o padrão XHTML foi criado e posteriormente atualizado para 1.1 – padrão este que se resume na HTML 4.01 com algumas variações em XML. A *World Wide Web* permaneceu praticamente estática neste período por várias razões.

Uma destas razões é que o W3C (órgão que regula os padrões Web, entre eles a HTML) tem um ciclo de versão demorado, que exige versões de rascunho e períodos de contribuição da comunidade, sugerindo novos conceitos e posteriormente validando, concordando e discordando dos mesmos – processos estes que levam tempo.

Em segundo lugar, os fabricantes de navegadores web demoravam muito a adotar o novo padrão, por muitas vezes adotando-o parcialmente – isso quando não fugiam dele, criando elementos e conceitos particulares, tornando-o um website aderente apenas a este ou aquele navegador web. Alguns fabricantes atualizavam seus navegadores web e padrões suportados apenas na troca do sistema operacional, o que poderia levar até três anos para acontecer. Isso sem falar nos navegadores do mundo mobile.

E, finalmente, como se os dois primeiros obstáculos não fossem o bastante, os internautas não atualizavam suas versões de navegador web – seja por inexperiência ou descaso – permanecendo com versões antigas que não davam suporte ao novo padrão. O absurdo chega ao ponto que o Microsoft Internet Explorer em sua versão 6.0 era o navegador web mais utilizado na Internet Brasileira, mais de 10 anos após seu lançamento. Esta incômoda liderança só foi vencida quando os websites mais acessados pelos brasileiros – o Facebook e o YouTube – passaram a barrar os usuários desta versão em particular de navegador.

Como resposta à lentidão do W3C, uma comunidade paralela formada por profissionais da Mozilla Foundation

### Fique por dentro

Depois de uma década de WWW basicamente estática e sem novidades tecnológicas, o mercado e a W3C trouxeram um grande conjunto de recursos e possibilidades nomeados HTML5. Muitas eram as deficiências e dificuldades enfrentadas pelos desenvolvedores web, e suas preces foram finalmente atendidas. Neste artigo apresentamos algumas tecnologias associadas à HTML5, bem como seus principais recursos, integrações, novas tags, e formas de efetuar a migração concisa das suas versões antigas para o novo padrão. Além disso, traremos também demonstrações simples do funcionamento de cada um dos recursos apresentados, constituindo, assim, um guia HTML5 completo para o desenvolvedor front-end.

(fabricante do Firefox), Opera Software (navegador Opera) e Apple (Safari) foi criada em 2004 com o nome de WHATWG (*Web Hypertext Application Technology Working Group*), com o intuito de discutir novos padrões e recursos para a Web.

Recebendo posteriormente contribuições de outras empresas como Google e Microsoft rapidamente o novo padrão foi ganhando forma de tal maneira que, três anos depois, foi submetido ao W3C que por sua vez decidiu adotá-lo batizando-o de HTML5. O novo padrão ajudou a enterrar o XHTML 2.0, padrão que o W3C estava trabalhando em 2007, considerado por muitos um equívoco – o padrão não era sequer compatível com a versão 1.1.

À primeira vista, a HTML5 parece se tratar meramente de um conjunto de novas *tags* para renderização de texto e formulários, impressão essa reforçada ao se folhear a maioria dos livros disponíveis hoje no mercado. Não se engane. Estas *tags* mencionadas são apenas a ponta do iceberg.

Embora seja chamada de HTML5 e a sigla signifique *HyperText Markup Language*, ou seja, linguagem de marcação de hipertexto, as novidades vão além disso. A HTML5 é um grande guarda-chuva tecnológico, pois pendurados a ele estão o novo padrão CSS na versão 3 e uma imensa gama de novas APIs na linguagem JavaScript estendendo as funcionalidades e possibilidades da WWW para patamar absolutamente impressionantes.

### Diferenças da HTML4

Este artigo não estaria completo se não discorrêssemos acerca das diferenças entre a nova HTML5 e a sua antecessora, a HTML4, esta que ocupou lugar de destaque no mundo web por muitos anos.

Ainda teremos um bom tempo de espera até que a HTML5 se torne de fato um padrão na web e que todos os browsers e tecnologias afins assumam a mesma como centro de implementação. Como programador web, é muito interessante que saiba quais as principais diferenças entre ambas as versões, justamente para que possa salvar tempo e aumentar a produtividade em situações como essa.

Uma das características mais marcantes dessa nova versão da linguagem é o fato de que a mesma não é uma versão final, isto é, estará sempre e continuamente mudando ao longo do tempo. Isso inclui dizer que os desenvolvedores da linguagem estarão sempre adicionando e removendo atributos, tags e o que considerarem interessantes à mesma. Ao mesmo tempo, constitui um risco se você estiver usando a mesma como algo definitivo no seu projeto. Isso significa que se optar pela HTML5 terá de seguir suas atualizações e estar constantemente evoluindo seu código também.

A HTML5 foi feita para ser simples, isso implica em uma sintaxe extremamente mais simples e limpa. A simples declaração do doctype foi apenas mais uma das facilidades incluídas na nova versão. Agora, você precisa inserir apenas um <!doctype html> no início do seu documento e tudo estará pronto. Além disso, a sintaxe da HTML5 é compatível também com a HTML4 e XHTML1.

A linguagem apresenta também um elemento novo, que veremos aqui no artigo, o <canvas>, responsável por substituir muitas das implementações antes feitas em Flash, o que faz muitos desenvolvedores considerar que este já se encontra obsoleto e futuramente morto.

A extensão de tags a um tool de novos e interessantes recursos fez uma grande diferença na linguagem. Tags como: <header> e <footer>, que estendem a funcionalidade de tabelas agora para a página como um todo, <section> e <article>, que permitem marcar áreas específicas dos layouts, <video> e <audio> para uma inclusão melhorada de conteúdos multimídia nas páginas, e <menu> e <figure> para bem arranjar textos, imagens e menus, trazem todo um conjunto de implementações e funcionalidades bem pertinentes para a web de hoje.

Além disso tudo, a remoção de alguns outros recursos como as tags <center>, <big>, <font>, etc fazem com a responsabilidade do CSS aliado à nova linguagem só aumente, otimizando o desenvolvimento front-end.

## Convertendo de versões antigas

Para converter códigos poliglotas e obsoletos para a nova versão da HTML5, basicamente temos de seguir três passos:

1. Remova os identificadores PUBLIC FPI e SYSTEM da declaração do DOCTYPE;
2. Substitua quaisquer tags depreciadas da HTML ou construa sua implementação baseada num código HTML que seja de acordo com os padrões da HTML5. Por exemplo, tenha certeza de garantir que qualquer conjunto de tags <col> para colunas de tabelas HTML sempre tenham um elemento “colgroup” como seu parente de configuração;

3. Comece a tirar vantagem dos novos recursos da HTML5, tais como convertendo suas div's para as tags de seção da HTML5.

Mas e se necessitarmos realizar conversões de versões mais antigas ainda da HTML? Esse tipo de conversão requer um pouco mais de trabalho e atenção, uma vez que ocorreram notórias mudanças nas versões da HTML entre os anos 1997 e 2000 com o objetivo de suportar conversores baseados em XML, dispositivos móveis com regras de conversão bem restritas, templates client-side cacheáveis além de agregação com outros tipos de conteúdo como os velhos feeds RSS. Vejamos mais alguns passos para tal:

1. Confira sempre se o documento inicia com uma declaração XML e uma declaração de DOCTYPE;
2. Verifique se o elemento de top <html> inclui o atributo “xmlns=”http://www.w3.org/1999/xhtml””. (A URI para XHTML e versões mais recentes da HTML incluem o ano “1999” porque foi definido aquele ano enquanto a W3C HTML Recommendation lançada em 2000 estava ainda em fase de desenvolvimento.);
3. Verifique se todas as tags estão marcadas com a tag de fim, ou se estão todas “self-closed” com o sinal de />. Não se esqueça de averiguar se os nomes dos elementos iniciam e terminam com a marcação em minúscula.
4. Verifique se todos os valores de atributos estão cercados pelas aspas. Verifique também se os atributos booleanos estão codificados na sua forma full, usando o atributo name entre aspas como o valor (attribute=”attribute”) quando o mesmo valor for true e o omitindo completamente quando o valor for false. A forma full será apropriadamente entendida pelos web browsers que convertem documentos com a ainda sintaxe HTML ou a sintaxe XML do HTML5. Evite usar formas minimizadas para selected=””, a qual o XPath trata como false ao invés de true.

Observe que a especificação HTML5 explicita estado como:  
*Os valores “true” e “false” não são permitidos em atributos booleanos*

Isso acontece porque os browsers olham para o código com o valor booleano para os atributos e tratarão a string “false” como um valor false enquanto browsers que somente olham para a presença ou ausência do atributo tratarão esse código como true, resultando em comportamentos inconsistentes e confusos.

Atributos booleanos que precisam ser mudados incluem alguns dos tipos ilustrados na **Tabela 1**.

Observe que “true” e “false” são valores válidos para alguns atributos “não booleanos”, em particular atributos enumerados que o atributo draggable.

Outra forma de se trabalhar migrando esse tipo de documento é verificando qual o tipo de versão que foi usada para construir o documento HTML. Uma boa forma de se fazer isso é submeter a URL do website ao site do W3C: Markup Validation Service (ver seção **Links**). Ao executar, os possíveis resultados são:

- HTML5: Indica que o site já foi convertido para o padrão HTML5. Por exemplo, o próprio site do Google é um bom exemplo de teste que usa o padrão.

- XHTML 1.0 Strict: Indica que o site está usando a versão padrão 2000 W3C da HTML. Por exemplo, o próprio site do W3C adere à essa versão.
- XHTML 1.0 Transitional or HTML 4.01 Transitional: Indica que o site está usando o formato transicional entre a versão padrão da HTML4 de 1997 e a versão padrão da HTML de 2000. Por exemplo, o site AltaVista.com usa o formato da HTML 4.01 Transitional, já o site da Microsoft usa XHTML 1.0 Transitional.
- HTML 4.01 Strict: Indica que o site está usando a versão antiga da HTML4 de 1997. Por exemplo, o site do Yahoo usa o antigo padrão.

Atributo	Mudança
Async	Mudar para <code>async="async"</code>
Checked	Mudar para <code>checked="checked"</code>
Compact	Mudar para <code>compact="compact"</code>
Declare	Mudar para <code>declare="declare"</code>
Defer	Mudar para <code>defer="defer"</code>
Disabled	Mudar para <code>disabled="disabled"</code>
Ismap	Mudar para <code>ismap="ismap"</code>
Multiple	Mudar para <code>multiple="multiple"</code>
Noresize	Mudar para <code>noresize="noresize"</code>
Noshade	Mudar para <code>noshade="noshade"</code>
Nowrap	Mudar para <code>nowrap="nowrap"</code>
Open	Mudar para <code>open="open"</code>
Readonly	Mudar para <code>readonly="readonly"</code>
Required	Mudar para <code>required="required"</code>
Reversed	Mudar para <code>reversed="reversed"</code>
Scoped	Mudar para <code>scoped="scoped"</code>
Selected	Mudar para <code>selected="selected"</code>

**Tabela 1.** Lista de atributos e respectivas mudanças de tipo booleano

## As oito áreas da HTML5

O novo padrão é vasto. Por esta razão, o W3C dividiu nestas novidades em oito áreas tecnológicas. A divisão ajuda na homologação do padrão e suporte dos fabricantes, que estão sendo feitos em porções. Estas áreas são:

- **Semantics (Semântica):** A ponta do iceberg. Nesta área, estão as novas tags que auxiliam na análise semântica dos textos presentes nas páginas. Muito úteis especialmente para melhorar a eficiência dos mecanismos de busca, como o Google. Adicionalmente, novas tags para renderização de formulários, em destaque caixas de texto com validação nativa.
- **Offline & Storage (Fora do Ar e Armazenamento):** Aborda todas as funcionalidades referentes ao tratamento do *website* quanto o visitante estiver em modo offline, além das novidades em armazenamento de informações no navegador como o LocalStorage e o Banco de Dados IndexedDB, indo mais além dos famigerados cookies.

• **Device Access (Acesso por Dispositivos):** Compreende todas as APIs que estendem a experiência de visitantes de dispositivos móveis como tablets e smartphones, como a Geolocalização (usando GPS ou outros recursos para determinar onde está o usuário), o uso do acelerômetro (utilizado para determinar a orientação do dispositivo no espaço), o uso do microfone e câmera destes dispositivos, além de eventos específicos para telas sensíveis a toque.

• **Connectivity (Conectividade):** Avanços incríveis na conectividade de aplicações web, como a comunicação usando Web-Sockets e SSE (*Server Side Events*), essenciais especialmente em jogos eletrônicos.

• **Multimedia:** Novas tags para a publicação de áudio e vídeo na Internet, além de novos formatos multimedia suportados.

• **3D, Graphics & Effects (3D, Gráficos e Efeitos):** Engloba aqui o Canvas que permite desenhar elementos gráficos em uma página, como uma tela de desenho. Adicionalmente, o suporte a WebGL para renderizações em 3D, o suporte ao formato SVG e efeitos 3D obtidos através de CSS3.

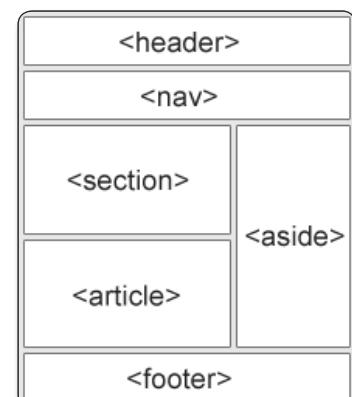
• **Performance & Integration (Performance & Integração):** Melhorias significativas na manipulação e submissão de formulários pela Internet. Possibilidades de processamento paralelo utilizando os novíssimos *Web Workers*, além de melhorias integração com o visitante (interface) com o excelente recurso de *Drag'n Drop* (arrastar-e-soltar).

• **CSS3:** Novíssimos estilos e efeitos sem sacrificar performance ou indexação dos mecanismos de busca.

A seguir, alguns exemplos do uso destes recursos.

## Semântica

Trata-se de um grande desafio para os mecanismos de busca da atualidade classificar a importância e relevância das informações contidas nas páginas. Com este intuito, novas tags foram criadas, dando maior importância semântica aos textos. Estas tags se comportam como a tradicional tag `<div>`, que leva este nome pois dividir o texto em blocos, até então anônimos. Agora os mesmos podem ser classificados, seguem as principais possibilidades (veja a **Figura 1**):



**Figura 1.** Exemplo de disposição das tags

- **<section>**: Apresenta a sessão, o tema da qual os artigos possuem em comum;
- **<nav>**: Trata-se dos elementos para a navegação do visitante, basicamente, o menu;
- **<article>**: Contém o texto do artigo, o conteúdo propriamente dito;
- **<aside>**: Informações relacionadas às informações do artigo, mas precisam ser separadas do contexto, como, por exemplo, notas de rodapé;
- **<header>**: Contém o cabeçalho;
- **<footer>**: Contém o rodapé do artigo;
- **<time>**: Armazena a data e hora daquela informação;
- **<mark>**: Serve para marcar parte do texto colocando-o em destaque ou para uma referência.

Observe a aplicabilidade das tags descritas na **Listagem 1**.

## Listagem 1. Exemplo de Código com as Tags de Semântica.

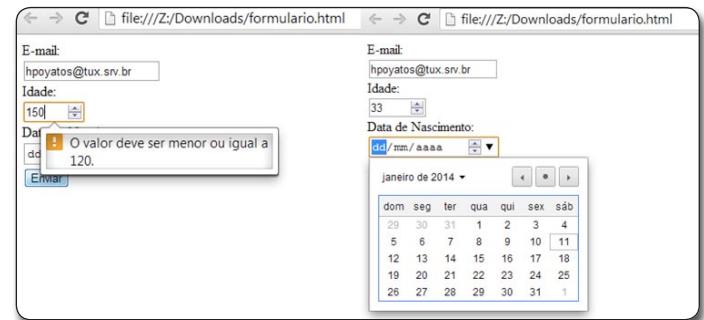
```
01 <header>Meus Artigos sobre HTML5</header>
02 <nav>
03   <a href="/html5/">HTML5</a> |
04   <a href="/css3/">CSS3</a> |
05   <a href="/js/">JavaScript</a>
06 </nav>
07 <section>Semântica</section>
08 <article>
09   <header>
10   <p class="post-date">12 de Janeiro de 2014</p>
11   <h1>Novas tags semânticas</h1>
12   </header>
13   <p>As novas tags semânticas já possuem um bom suporte pelos
14     navegadores web no mercado, entre eles o Chrome, um dos navegadores
15     mais utilizados.</p>
16 </article>
17 <aside>
18   <h1>Google Chrome</h1>
19   <p>Navegador web desenvolvido pela Google.</p>
20 </aside>
21 <footer>
22   <p>Postado por: Henrique Poyatos</p>
23   <p><time pubdate datetime="2014-01-10"></time></p>
24 </footer>
```

Quando tratamos do uso de formulário, a tag **<input>** já é amplamente atualizada para caixas de texto, botões de seleção e de acionamento. Entretanto, quando fosse necessário validar uma informação em uma caixa de texto, longos trechos de codificação JavaScript se faziam necessários para cumprir tal objetivo.

Destacam-se, portanto, alguns tipos novos e interessantes para a tag **<input>**. Vários foram criados e estes já possuem validação nativa por parte do navegador web. Um exemplo é apresentado na **Listagem 2**.

O exemplo utiliza os tipos “email”, “number” e “date”, embora existam outros. Repare também nos novos atributos **placeholder** e **required**. O **placeholder** é responsável por exibir mensagens como marca d’água na caixa de texto que se encontra vazia. Já o **required** (linha 11) que dizer obrigatório, ou seja, exige o preenchimento da caixa de texto, impedindo o formulário de ser submetido enquanto a condição não for satisfeita.

O exemplo faz uso também dos novos atributos **min** e **max** (linha 13), úteis para se estabelecer intervalos números para o tipo “number”. Veja a **Figura 2**.



**Figura 2.** Exemplo da **Listagem 2** em execução. Novos tipos de caixa de texto validadas nativamente

## Listagem 2. Exemplo de Formulário com tipos novos de **<input>**

```
01 <!DOCTYPE HTML>
02 <html lang="pt-br">
03 <head>
04   <meta charset="UTF-8">
05   <title>Exemplo de Formulário</title>
06 </head>
07 <body>
08   <form method="POST">
09     <label for="endemail">E-mail: </label><br />
10     <input type="email" name="endemail" placeholder="Digite seu E-mail"
11       required /><br />
12     <label for="idade">Idade: </label><br />
13     <input type="number" name="idade" min=0 max=120
14       placeholder="Informe sua Idade"/><br />
15     <label for="datNasc">Data de Nascimento: </label><br />
16     <input type="date" name="datNasc"/>
17     <br />
18     <input type="submit" value="Enviar" />
19   </form>
20 </body>
21 </html>
```

Como já dito anteriormente, as novidades e recursos desta primeira área tecnológica são amplamente divulgados em blogs e livros sobre HTML5. Nas próximas áreas, iremos nos aprofundar em recursos menos divulgados, mas igualmente interessantes e inovadores.

## Offline & Storage

Não existe nada tão desagradável para um usuário do que uma brusca queda de conexão durante a navegação, especialmente o preenchimento de um grande formulário de dados. A necessidade de se recomeçar o procedimento pode levar até mesmo o não retorno do internauta à aplicação. Além disso, existem procedimentos como a leitura de páginas institucionais, por exemplo, sequer necessitariam de uma conexão permanente.

Para tais situações, o W3C trouxe uma solução simples porém eficiente: o tratamento off-line para o site. Isso significa que, caso a conexão caia durante o uso da aplicação, tomados os devidos

cuidados, ela não vai parar de funcionar – desde que, é claro, não sejam necessárias informações adicionais.

O tratamento off-line é configurado em um arquivo de manifesto — que não é nada mais é do que uma lista de arquivos necessários para que a aplicação funcione sem conexão. Assim, os arquivos são automaticamente copiados em cache pelo navegador, com a finalidade de mantê-la funcionando.

Além disso, o manifesto ainda pode conter uma relação dos arquivos não necessários para uso offline (dos quais não serão armazenados localmente) e uma listagem de arquivos alternativos para uso offline – ou seja, devidamente preparados para este fim – que tomarão o lugar de outros quando a conexão cair. Observe a **Listagem 3**.

**Listagem 3.** Exemplo de arquivo de manifesto.

```
01 CACHE MANIFEST
02
03 CACHE:
04 images/favicon.ico
05 images/logo.png
06 index.html
07 css/stylesheet.css
08 js/script.js
09
10 NETWORK:
11 login.php
12 /application
13
14 Fallback:
15 /index.php /index_offline.html
16 *.php /offline.html
```

Na seção CACHE relacionaremos todos os arquivos a serem copiados pelo navegador para que estejam disponíveis durante a queda de conexão. Repare que devemos no ater a arquivos como .html, .js, .css e imagens como .png, .jpg e .gif que são passíveis de rodar no cliente (o chamado *run at client*). Arquivos que necessitem de interpretador do lado servidor (como .php, .aspx, .jsp) precisarão ser substituídos no fallback.

Na seção NETWORK, relate os arquivos não necessários para uso offline. E finalmente em Fallback, relacionamos quais arquivos serão substituídos por outros (relacionados na sequência, separados por espaço).

Armazenar informações de aplicações é uma necessidade recorrente que os bancos de dados com seus excelentes mecanismos de persistência, cumprem de forma rápida e segura. Entretanto, nem sempre o banco de dados se faz necessário, pois algumas informações dizem respeito a utilização da aplicação naquele momento – útil para a interface do usuário, mas resulta em um tráfego de rede desnecessário.

Neste caso, guardamos as informações no lado cliente e, por décadas, a única forma de se fazer isso era o cookie. Infelizmente, este recurso se tornou cada vez mais obsoleto, pois guarda um volume de informações muito pequeno (até 4 KBytes) e de uma forma muito insegura.

Para resolver o problema, os desenvolvedores da HTML5 criaram duas novas possibilidades para armazenamentos em navegadores.

A primeira delas é o Banco de Dados do lado cliente, antigamente representado pelo Web SQL Database, descontinuado e agora substituído pelo IndexedDB.

O banco de dados IndexedDB permite o armazenamento e manipulação de um grande volume de informações usando linguagem SQL ou outras técnicas, e por esta razão ele é considerado um banco noSQL (not only SQL).

O exemplo da **Listagem 4** cria um repositório de produtos, populando a tabela com três deles.

**Listagem 4.** Exemplo de armazenamento utilizando IndexedDB

```
01 <html>
02   <head>
03     <meta charset="utf-8">
04     <title>Exemplo IndexedDB</title>
05   </head>
06   <body>
07   </body>
08 </html>
09 <script type="text/javascript">
10 var request = indexedDB.open("lojinha");
11 request.onupgradeneeded = function()
12 {
13   // Se o banco de dados não existir ainda, cria objetos de armazenamento
14   var db = request.result;
15   var store = db.createObjectStore("produtos", {keyPath: "codigo"});
16   var nomeIdx = store.createIndex("porNome", "titulo", {unique: true});
17   var fabricanteIdx = store.createIndex("porFabricante", "fabricante");
18   // Populando o banco com alguns produtos
19   store.put({codigo: 1,
20             nome: "DVD - Batman O Cavaleiro das Trevas - A Trilogia",
21             fabricante: "Warner Bros",
22             preco: 39.90});
23   store.put({codigo: 2,
24             nome: "Blu-ray - O Homem de Aço",
25             fabricante: "Warner Bros",
26             preco: 69.90});
27   store.put({codigo: 3,
28             nome: "DVD - Wolverine Imortal",
29             fabricante: "Fox",
30             preco: 19.90});
31
32 };
33 request.onsuccess = function() {
34   db = request.result;
35 };
36 </script>
```

Teremos como resultado a tela presente na **Figura 3**.

A outra possibilidade de armazenamento local são os chamados sessionStorage e localStorage. São recursos mais simples que o IndexedDB mas mais robustos do que os antigos cookies. O funcionamento de ambos é similar, a única diferença é que armazenando utilizando sessionStorage compartilha as informações do website todo, mesmo abrindo várias abas – diferente do localStorage, cujas informações ficam restritas à apenas a aba em funcionamento.

Na **Listagem 5** é exposto um exemplo de funcionamento. Como resultado temos a **Figura 4**.

# Programando em HTML5



Figura 3. Produtos sendo armazenados conforme Listagem 4

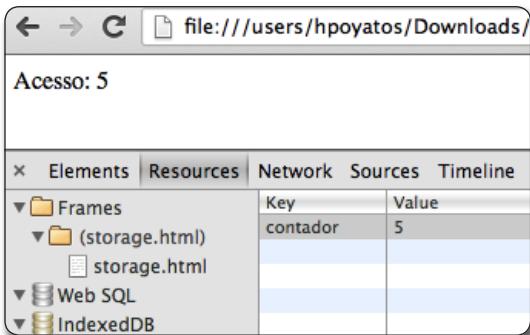


Figura 4. Armazenamento de informações usando sessionStorage conforme Listagem 5.

Listagem 5. Exemplo de armazenamento usando SessionStorage.

```
01 <html>
02 <head>
03   <meta charset="utf-8">
04   <title>Exemplo de Armazenagem</title>
05 </head>
06 <body>
07   <div id="contador"></div>
08 </body>
09 </html>
10 <script type="text/javascript">
11 // Verifica se existe suporte ao sessionStorage
12 if(sessionStorage)
13 {
14   if(!sessionStorage.contador) sessionStorage.contador = 0;
15   sessionStorage.contador++;
16   document.getElementById('contador').innerHTML ='Acesso:
17   '+sessionStorage.contador;
18 }
19 </script>
```

## Recursos para Dispositivos Móveis

Nossas vidas mudaram completamente depois da invenção dos smartphones e tablets. Avanços na usabilidade com suas telas sensíveis a toque, capacidades de processamento espantosas para dispositivos tão pequenos e, é claro, seus recursos úteis como acesso ao sistema de GPS e o acelerômetro.

Quanto ao GPS, este conjunto de satélites que triangula fora de órbita nossos sinais fornecendo latitude e longitude, transformou-se na forma em que nos locomovemos em nossas cidades e até mesmo como procuramos produtos e serviços.

Na ausência de um dispositivo que acesse tais satélites, quaisquer sinais modernos de onda modernos (como Bluetooth, Wi-Fi, GSM, CDMA, entre outros) podem ajudar a cumprir tal função, mesmo que de forma aproximada.

Com um recurso tão rico à disposição, os desenvolvedores da HTML5 criaram uma API para que o browser solicite tal recurso ao sistema operacional, que por sua vez fará uso do hardware buscando tal informação. Veja na Listagem 6 um exemplo simples do uso deste recurso.

Obtendo tal resultado prático, temos o resultado exposto na Figura 5.

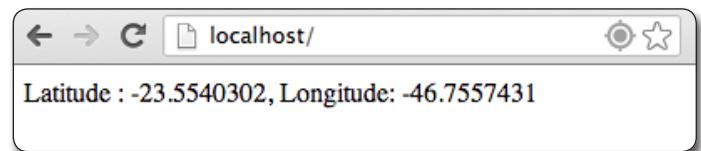


Figura 5. Latitude e Longitude utilizando Geolocalização apresentado na Listagem 6

Listagem 6. Exemplo de Geolocalização.

```
01 <html>
02 <head>
03   <meta charset="utf-8">
04   <title>Exemplo Geolocalização</title>
05 </head>
06 <body>
07   <div id="coordenadas"></div>
08 </body>
09 </html>
10
11 <script type="text/javascript">
12 // Verificando se existe suporte a geolocalização
13 if(navigator.geolocation)
14 {
15   // Método getCurrentPosition() retorna
16   // objeto com atributos coords (coordenadas)
17   navigator.geolocation.getCurrentPosition(resultado);
18   function resultado(posicao)
19   {
20     // Objeto posição com propriedade coords
21     document.getElementById('coordenadas').innerHTML =
22     'Latitude:' + posicao.coords.latitude + ', Longitude:
23     ' + posicao.coords.longitude;
24   }
25 </script>
```

Claro que com tais coordenadas e um sistema de mapas preciso (como o Google Maps) poderíamos apresentar a informação de uma forma muito mais visual.

Conforme dito, a mudança na acessibilidade trazida com os dispositivos móveis provoca uma quebra de paradigma: sua principal interface com o usuário é uma tela sensível a toque. Como interagir por esta interface, uma vez que o tradicional JavaScript possuía apenas eventos disparados por teclado e mouse? Foram então criados eventos disparados pelo toque, ou Touch Events.

Na sequência, uma demonstração de seu uso na Listagem 7 e o resultado na Figura 6.



**Figura 6.** Tela sensível a toque conforme apresentado na **Listagem 7**

#### Listagem 7. Exemplo de Evento de Toque.

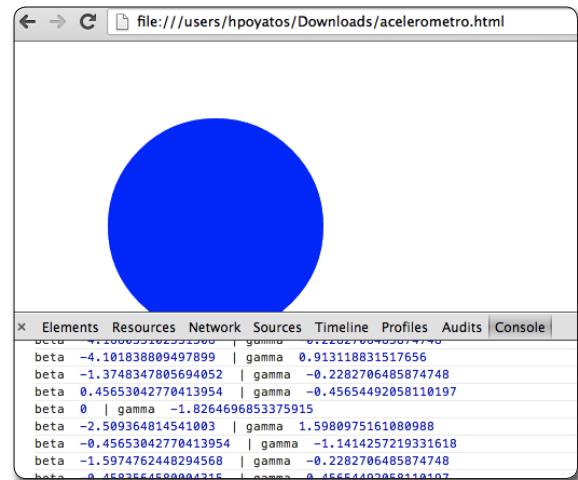
```

01 <html>
02 <head>
03   <meta charset="utf-8">
04   <title>Exemplo com Toques</title>
05 </head>
06 <style>
07 #quadrado {
08   width: 600px;
09   height: 370px;
10   border: 1px solid blue;
11   background-color: blue;
12   color: white;
13   text-align: center;
14   font-size: 40px;
15   padding-top: 130px;
16 }
17 </style>
18 <body>
19   <div id='quadrado'></div>
20 </body>
21 </html>
22 <script type="text/javascript">
23 // Evento touchstart começa assim que a tela é tocada
24 document.getElementById('quadrado').addEventListener('touchstart',
25 function(evento){
26   // targetTouches é uma lista de toques sobre a superfície toda
27   if (evento.touches.item(0) == evento.targetTouches.item(0)) {
28     document.getElementById('quadrado').innerHTML = "TOQUE
29     DETECTADO!";
30   }
31 }, false);
32 </script>
```

E finalmente, mas não menos importante, a possibilidade do uso do acelerômetro, um recurso presente nos dispositivos portáteis (e alguns notebooks também) que permite determinar se o aparelho está sendo rotacionado de alguma maneira. O Evento de Device Orientation possui atributos conhecidos como alpha, beta e gamma, que retornam números que representam os movimentos de rotação.

Veja um exemplo de uso na **Listagem 8**.

Teremos como resultado o conteúdo da **Figura 7**.



**Figura 7.** Esfera feita em CSS se movendo pela tela com acelerômetro na **Listagem 8**

#### Listagem 8. Exemplo de Uso do Acelerômetro.

```

01 <!DOCTYPE HTML>
02 <html lang="pt-br">
03 <head>
04   <meta charset="UTF-8">
05   <title>Exemplo Acelerômetro</title>
06 </head>
07 <style>
08 #circulo {
09   width: 200px; height: 200px;
10   border-radius: 200px;
11   background-color: blue;
12   position: relative;
13 }
14 </style>
15 <body>
16   <div id="circulo"></div>
17 </body>
18 </html>
19 <script type="text/JavaScript">
20 //Verifica se o dispositivo+navegador possui suporte a acelerômetro
21 if (window.DeviceOrientationEvent) {
22   //Aplica o evento deviceorientation que será na função move()
23   window.addEventListener("deviceorientation", move, true);
24   function move(evento) {
25     // Mostra coordenadas beta e gamma no console
26     console.log('beta', evento.beta, '| gamma', evento.gamma)
27     if (evento.gamma > 0) {
28       document.getElementById('circulo').style.left =
29         (document.getElementById('circulo').offsetLeft + 1) + 'px';
30     } else {
31       document.getElementById('circulo').style.left =
32         (document.getElementById('circulo').offsetLeft - 20) + 'px';
33     }
34     if (evento.beta > 0) {
35       document.getElementById('circulo').style.top =
36         (document.getElementById('circulo').offsetTop + 1) + 'px';
37     } else {
38       document.getElementById('circulo').style.top =
39         (document.getElementById('circulo').offsetTop - 10) + 'px';
40     }
41   }
42 }
43 </script>
```

## Conectividade

Os novos recursos da área de conectividade são os mais impressionantes e também os mais subestimados. Novas abordagens como os Web Sockets e o SSE trazem inovações essenciais, especialmente aos desenvolvedores de jogos eletrônicos.

Para se entender a importância dos Web Sockets, vamos entender primeiramente como o protocolo HTTP funciona: o usuário (lado cliente) interage com a interface e esta interação dispara uma requisição para o servidor, que processa a requisição e a devolve ao solicitante, encerrando a comunicação. Cada nova interação dispara novas requisições que são abertas e fechadas, ou seja, é seguro afirmar que o servidor reage a requisições, emitindo respostas a partir delas. Também podemos afirmar que a requisição sempre parte no sentido do cliente para o servidor, sendo, portanto, de inicialização unilateral, já que o servidor nunca pode começar a comunicação.

Sendo assim, caso precisemos enviar novas informações regularmente no sentido inverso (do servidor para o cliente), o cliente, por sua vez, deve possuir rotinas que regularmente realizam requisições ao servidor em busca de novas informações, procedimento conhecido como polling. Entretanto, esta abordagem aumenta a latência, sobrecarregando a conexão com inúmeras requisições que na maioria dos casos serão desnecessárias.

Para solucionar o problema, os desenvolvedores da HTML5 trouxeram a tecnologia de sockets criada décadas atrás no Unix – trata-se de um canal de comunicação que não usa o protocolo HTTP e sim um protocolo persistente cuja comunicação pode ser iniciada de ambos os lados (full-duplex).

A solução também se torna um obstáculo. Abrir uma nova porta de serviço de rede em padrão socket exige certos privilégios de segurança que os servidores de hospedagem, que tradicionalmente trabalham com servidores compartilhados por vários clientes, não colocam à disposição de seus clientes. A estratégia fica então restrita para projetos que utilizem infraestrutura própria e dedicada, ou um serviço de *Cloud Computing*.

Conforme dito antes, Web Sockets não é a única novidade na área de conectividade. Baseado em protocolo HTTP, outra boa novidade são os *Server Side Events* (SSEs) que quer dizer, em tradução livre, eventos do lado do servidor. Ele cria um canal de comunicação simples e enxuto buscando por informações novas sempre que necessário. Trata-se sim do *polling* mencionado anteriormente, mas leve e enxuto, pois é controlado nativamente pelo navegador, substituindo linhas e mais linhas de JavaScripts e AJAXs.

Vamos a um exemplo de uso do SSE. A **Listagem 9** está em código PHP e deve ser armazenada no servidor. Trata-se de um código muito simples, mesmo para aqueles que não programam em PHP: o código sorteia entre três cores (vermelho, azul e verde) e exibe uma informação no formato pré-determinado pelo SSE: data: red/blue/green(dois pular linhas).

O código da **Listagem 10** é o JavaScript acessando a API de SSE, recebendo a informação do servidor (<http://localhost/cor.php>) e usando a cor sorteada como cor para plano de fundo.

**Listagem 9.** Arquivo cor.php. Sorteia randomicamente entre três cores e exibe a informação em um formato pré-determinado.

```
01 <?php
02 header('Content-Type: text/event-stream; charset=utf-8');
03 header('Cache-Control: no-cache');
04
05 switch(rand(1, 3))
06 {
07 case 1:
08 $cor = "red";
09 break;
10 case 2:
11 $cor = "blue";
12 break;
13 case 3:
14 $cor = "green";
15 break;
16 }
17 echo "data: \"$cor\"\n\n";
18 ob_flush();
19 flush();
20 ?>
```

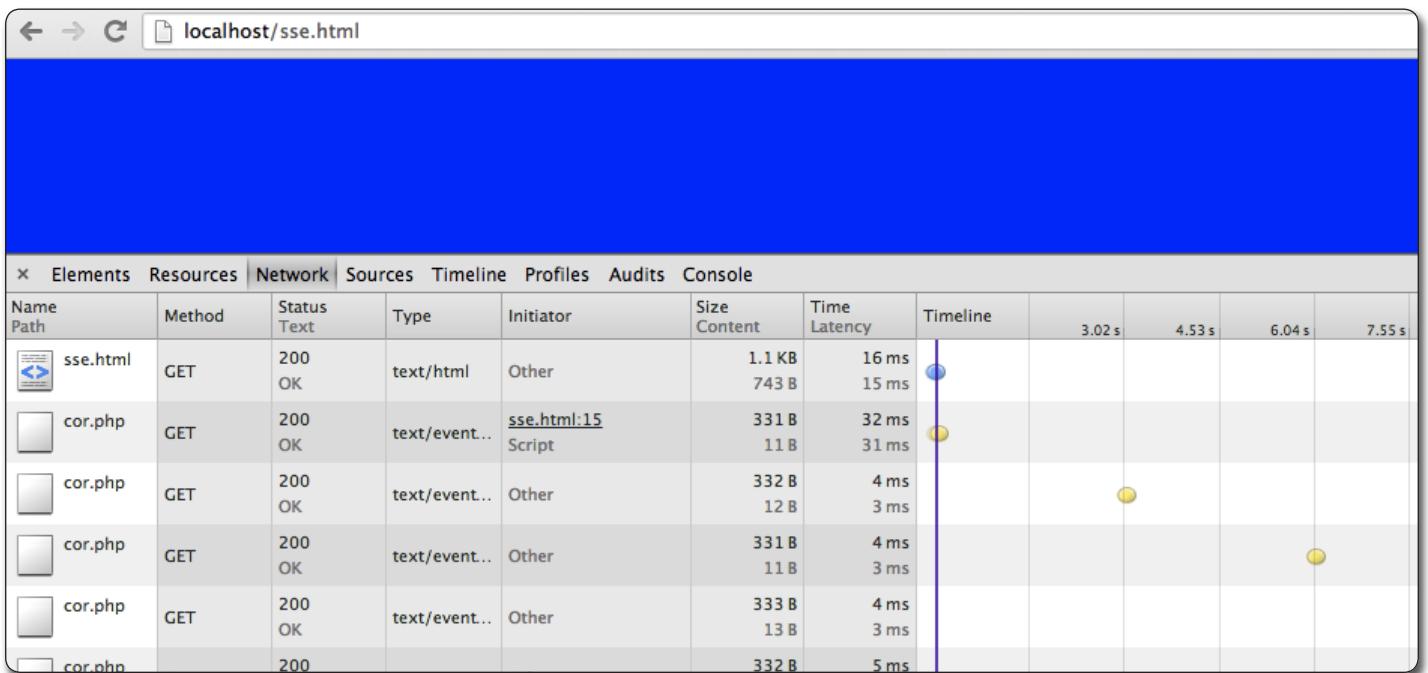
**Listagem 10.** Código exemplo utilizando SSE.

```
01 <!DOCTYPE HTML>
02 <html lang="pt-br">
03 <head>
04 <meta charset="UTF-8">
05 <title>Server Side Events - Exemplo Simples</title>
06 </head>
07 <body id="corpo">
08 </body>
09 </html>
10 <script type="text/JavaScript">
11 //Verificando se há suporte para Server Side Events
12 if(typeof(EventSource) !== "undefined")
13 {
14 // Chamada do Server Side Event
15 var servidor = new EventSource("http://localhost/cor.php");
16 // Utilizando o evento onMessage - tratamento do que será feito
17 // sempre que um "data" chegar.
18 servidor.onmessage = function(event)
19 {
20 document.getElementById("corpo").style.backgroundColor = event.data;
21 }
22 }
23 else
24 {
25 document.getElementById("corpo").innerHTML =
26 "Este navegador não possui 26 suporte à Server Side Events";
27 }
28 </script>
```

Obtemos o seguinte resultado exposto na **Figura 8**.

## Gráficos e Efeitos em 3D

Uma grande novidade para a HTML5 é o recurso conhecido como Canvas. Equivalente a uma tela de pintura tradicional, ela permite, de forma bem simples, desenhar elementos “on the fly” (em tempo de execução). A área de desenho é apresentada pela tag <canvas>, enquanto o trabalho artístico fica a cargo de comandos utilizando sua API em *JavaScript*. A **Listagem 11** traz uma demonstração do que podemos fazer. O resultado pode ser observado na **Figura 9**.



**Figura 8.** Exemplo SEE das **Listagens 9 e 10** em execução – Cores de fundo do documento trocadas regularmente pelo servidor usando SSE

#### **Listagem 11.** Exemplo de Código usando HTML5 Canvas.

```

01 <!DOCTYPE html>
02 <html>
03 <meta charset="utf-8">
04 <body>
05   <canvas id="telaEmBranco" width="400" height="300"></canvas>
06 </body>
07 </html>
08
09 <script type="text/javascript">
10   var canvas=document.getElementById("telaEmBranco");
11   var ctx=canvas.getContext("2d");
12   ctx.beginPath();
13   /*
14   Definindo um retângulo
15   Os dois primeiros números são coordenadas X e Y iniciais,
16   os seguintes são coordenadas finais
17   */
18   ctx.rect(0,0,400,200);
19   //Desenhando o retângulo
20   ctx.stroke();
21
22 //Limpa para novo elemento gráfico.
23 ctx.beginPath();
24 //Definindo um arco.
25 ctx.arc(200,100,60,50,2*Math.PI);
26 //Definindo o preenchimento dele em vermelho.
27 ctx.fillStyle="red";
28 //Desenhando e preenchendo.
29 ctx.fill();
30
31 ctx.beginPath();
32 ctx.font="30px Arial";
33 // Aplicando o texto na imagem
34 ctx.fillText("Bandeira do Japão",70,250);
35 </script>
```



**Bandeira do Japão**

**Figura 9.** Desenho em Canvas apresentado na **Listagem 11**

#### Integração

Um dos grandes problemas no envio de arquivos pela web (*upload*) foi finalmente resolvido com o chamado XMLHttpRequest2: ao se enviar arquivos muito grandes, não se tinha uma ideia de progresso deste envio.

O mérito é de um novo evento incluído na API, onprogress, incluído no atributo upload. O mesmo recebe um objeto que possui o atributo lengthComputable, que traz informações de progresso que podem ser exibidas em uma barra de progresso (novidade também a tag <progress>, específica para este fim). Vejamos um exemplo em funcionamento na **Listagem 12**.

Ela traz um pequeno trecho em PHP, necessário para receber o arquivo enviado. Na sequência, o código da **Listagem 13** apresenta o JavaScript que manipula e gerencia o upload.

# Programando em HTML5

O exemplo demonstra algumas novidades: o uso de FormData, a possibilidade de se criar ou complementar formulários de maneira muito mais fácil e simples. Além disso, o uso do XMLHttpRequest 2, com o atributo upload, específico para este tipo de operação, que retorna em seu evento onprogress um objeto com atributos lengthComputable, loaded e total, que trazem informações relevantes da operação, entre elas suas parciais. Como resultado, temos a imagem apresentada na **Figura 10**.

A integração homem-máquina também recebeu novidades. O recurso de Drag'n Drop (arrastar-e-soltar) tão utilizado em interfaces gráficas dos sistemas operacionais, agora pode ser utilizado em páginas e aplicações web. Um elemento pode ser considerado dragable ("arrastável"), podendo até mesmo transportar informações contidas nele para outros elementos (que chamaremos aqui de "alvo").

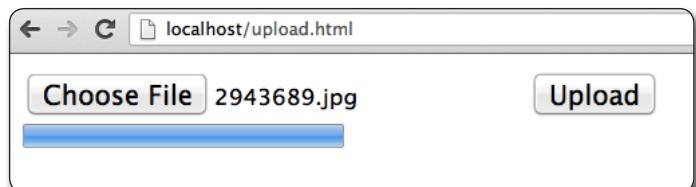
**Listagem 12.** Código upload.php para receber o arquivo enviado.

```
01 <?php  
02 move_uploaded_file($_FILES['arquivo']['tmp_name'], $_FILES['arquivo'][  
03 'name']);  
04 ?>
```

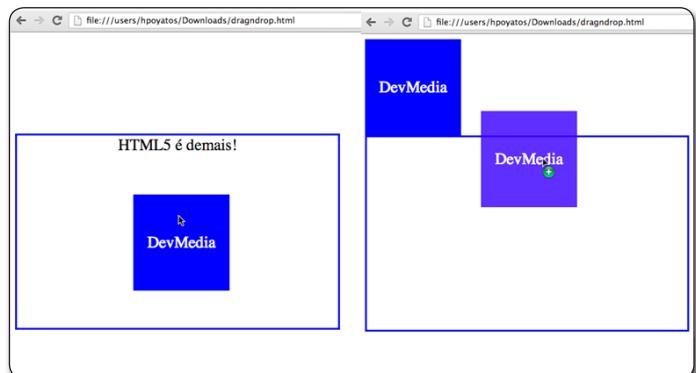
**Listagem 13.** Código exemplo para Upload com barra de progresso.

```
01 <!DOCTYPE HTML>  
02 <html lang="pt-br">  
03 <head>  
04 <meta charset="UTF-8">  
05 <title>Exemplo de Upload</title>  
06 </head>  
07 <body>  
08 <form id="formulario_upload">  
09 <input type="file" name="arquivo" id="arquivo"/>  
10 <input type="button" id="botao_upload" value="Upload"/>  
11 </form>  
12 <progress min="0" max="100" value="0">0% complete</progress>  
13 </body>  
14 </html>  
15 <script type="text/JavaScript">  
16 var botao = document.getElementById("botao_upload");  
17  
18 botao.onclick = function()  
19 {  
20 var xhr2 = new XMLHttpRequest();  
21 xhr2.open('POST', 'upload.php', true);  
22 var arquivoInput = document.getElementById('arquivo');  
23 //files[0] o elemento pode receber de um arquivo  
24 var arquivo = arquivoInput.files[0];  
25 // É carregado em um formulário FormData(), outra novidade  
26 var formulario = new FormData();  
27 formulario.append('arquivo', arquivo);  
28 var progressBar = document.querySelector('progress');  
29 xhr2.upload.onprogress = function(e)  
30 {  
31 //Verifica se o atributo lengthComputable existe  
32 if (e.lengthComputable)  
33 {  
34 // Calcula percentual e carrega <progress>  
35 progressBar.value = (e.loaded / e.total) * 100;  
36 }  
37 };  
38 // Envia o formulário com o arquivo anexo.  
39 xhr2.send(formulario);  
40 };  
41 </script>
```

Vejamos um exemplo deste recurso na **Listagem 14** e a execução na **Figura 11**.



**Figura 10.** Upload com barra de progresso conforme **Listagens 12 e 13**



**Figura 11.** Recurso de Drag'n Drop apresentado na **Listagem 14** em funcionamento

Muitos são os recursos trazidos pela tecnologia HTML5 e pela nova geração de navegadores web e, portanto, muitas são as oportunidades de seu uso, que podem ir desde experiências mais ricas de navegação e interatividade em um website até avançadíssimos jogos eletrônicos baseados em web. Animações e streaming de áudio e vídeo, antes possíveis apenas pelo plugin proprietário Adobe Flash, agora são realidade de forma nativa.

Tratando em particular dos jogos eletrônicos, WebSockets ou SSE trazem uma troca de informações ágil, essencial para partidas com múltiplos jogadores. Animações realizadas usando Canvas, WebGL ou recursos do CSS3, acompanhados por uma envolvente trilha sonora reproduzida pelo novo suporte de streaming de áudio, utilizando-se dos eventos de toque em tela, acelerômetro e recurso de *Drag'n Drop* farão os jogos eletrônicos baseados em web alcançarem patamares possíveis outrora apenas por jogos instaláveis em sistemas operacionais.

Fallbacks e condicionais verificando a existência da API (como em muitos exemplos deste artigo) farão com que até mesmo visitantes oriundos de navegadores mais antigos possam visitar sua página ou aplicação, embora não tenham a mesma experiência. Frameworks como o Modernizr auxiliam nos fallbacks e podem providenciar polyfills, que são a possibilidade de preencher a falta de suporte a alguns recursos da HTML5 por recursos mais抗igos ou plugins.

Os novos recursos podem ser muito úteis em Layout Responsivo – trata-se de criar versões do website planejadas para vários tamanhos de tela e processadores, ou seja, em um único código criar páginas que se adaptem às telas que vão desde pequenos smartphones até televisores de muitas polegadas.

#### Listagem 14. Exemplo de Recurso de Drag 'n Drop.

```
01 <!DOCTYPE HTML>
02 <html lang="pt-br">
03 <head>
04 <meta charset="UTF-8">
05 <title>Exemplo Drag and Drop</title>
06 <style>
07   #arrastavel
08  {
09    position: relative; width:150px; height:90px;
10   background-color:blue;
11   text-align: center; font-size: 25px; color: white;
12   padding-top: 60px;
13 }
14
15 #alvo
16 {
17   position: relative; width:500px; height:300px;
18   border-color:black; border-style:solid;
19   font-size: 25px; text-align: center;
20 }
21 </style>
22 </head>
23 <body>
24 <!-- Elemento arrastável -->
25 <div id="arrastavel" draggable="true">DevMedia</div>
26 <!-- Elemento Alvo -->
27 <div id="alvo"></div>
28 </body>
29 </html>
30 <script type="text/javascript">
31 //Elemento arrastável
32 var arrastavel = document.getElementById("arrastavel");
33 // Evento ondragstart - Posso definir informações a serem transferidas
34 arrastavel.ondragstart = function(event) {
35   event.dataTransfer.setData("Info","HTML5 é demais!");
36 }
37 // Evento ondragend - Ao terminar de arrastar
38 //mudar o posicionamento do objeto
39 arrastavel.ondragend = function(event){
40   this.style.left = event.pageX + "px";
41   this.style.top = event.pageY-150 + "px";
42 }
43 //Elemento alvo
44 var alvo = document.getElementById("alvo");
45
46 //Evento ondragenter - pinta a borda do alvo de azul ao adentrar
47 alvo.ondragenter = function(event) {
48   this.style.borderColor = "blue";
49 }
50 //Evento ondragleave - devolve a coloração original da borda a sair
51 alvo.ondragleave = function(event) {
52   this.style.borderColor = "";
53 }
54 //Evento ondragover - preventDefault()
55 alvo.ondragover = function(event) {
56   event.preventDefault();
57 }
58 //Evento ondrop - Transfere a informação para dentro do alvo
59 alvo.ondrop = function(event) {
60   this.innerHTML = event.dataTransfer.getData("Info");
61 }
62 </script>
```

#### Autor



##### Henrique Poyatos

[hpoyatos@tux.srv.br](mailto:hpoyatos@tux.srv.br)



Formado em Tecnologia em Processamento de Dados e pós-graduado em Gerenciamento de Projetos pela FIAP. Atua no mercado de desenvolvimento para Internet desde 1996, coordenou projetos de tecnologia para empresas como Caixa Econômica Federal, Nossa Caixa, Mc Donald's, Metrô de São Paulo e projetos para clientes internacionais. Atua hoje como professor nas faculdades FIAP e BandTec e é instrutor da Impacta.

#### Links:

##### As 8 áreas da HTML5

<http://www.w3.org/html/logo/#the-technology>

##### Can I Use?

<http://caniuse.com/>

##### Site do Modernizr

<http://www.modernizr.com/>

# Metro: Criando um Menu Metro com HTML e CSS

## Crie interfaces responsivas e menus semelhantes ao Windows 8

O chamado estilo METRO é o novo padrão de interface das aplicações da Microsoft, podendo ser visto, por exemplo, no Windows Phone 7.5, Windows Phone 8 e Windows 8, tendo os dois últimos sido lançados no ano de 2012. O menu principal destes softwares possui um formato simples, prezando pela praticidade e leveza, mas sem deixar de lado a importância de um layout agradável, fator que tem se tornado relevante no desenvolvimento de sistemas, pois os usuários buscam cada vez mais interfaces elegantes e atualizadas.

Essa nova interface (do menu) é baseada em tiles (blocos) de, geralmente, dois tamanhos principais (um quadrado e um retangular) e várias cores, que facilitam o acesso ao menu em dispositivos com tecnologia touch screen. Nesse tipo de dispositivo, como os usuários utilizam geralmente os dedos para acessar as aplicações e manipulá-las, assim como as telas geralmente possuem dimensões mais reduzidas quando comparadas aos monitores desktop, os elementos gráficos da janela devem ter dimensões e formatos razoáveis para facilitar essas tarefas.

### Observação

É importante ressaltar que o estilo Metro não se resume a esse menu sobre o qual foi falado, existe um novo “grupo” de aplicações para Windows 8 e Windows Phone chamadas de Metro Style Apps cuja interface é baseada nesse padrão. Porém, como o objetivo desse artigo é desenvolver um menu semelhante ao menu Iniciar do Windows 8, é relevante falar apenas sobre sua estrutura e funcionamento.

Um exemplo deste menu pode ser visto na **Figura 1** a seguir, onde se observa o menu iniciar do Windows 8.

Apesar de esse modelo ser mais utilizado em aplicações desktop e mobile, será apresentada aqui uma solução para o desenvolvimento de uma interface semelhante para páginas web, utilizando apenas as Web Standards.

### Fique por dentro

Um dos grandes e mais recentes adventos da programação front-end se chama: responsividade. Em vista disso, é cada vez mais comum a existência de aplicações web que se adequem automaticamente aos diferentes tipos de telas, e, paralelamente, os desenvolvedores necessitam aprender técnicas e formas de implementar seus websites de forma a se encaixar nesse conceito.

Este artigo apresenta uma forma simples, porém eficiente, de desenvolvimento de um menu semelhante à tela inicial do Windows 8, utilizando apenas as linguagens muito comumente chamadas de Web Standards (Padrões Web), tais como HTML, CSS e JavaScript. Será desenvolvida uma interface responsiva baseada em tiles (blocos), que apesar de visualmente simples, como se vê no mais recente sistema operacional da Microsoft, é bastante adequada a dispositivos touch screen, facilitando o acesso aos itens do menu.



**Figura 1.** Menu inicial do Windows 8 no estilo metro

Neste artigo, porém, não serão desenvolvidas funcionalidades mais complexas como a movimentação de tiles. O foco será a estrutura geral que poderá ser customizada e aprimorada pelo leitor, desde que o mesmo tenha conhecimentos nas tecnologias utilizadas.

## Como funcionará o menu

Conforme dito anteriormente, o layout do menu é bastante simples. A estrutura geral é organizada em linhas (pode-se também considerar a divisão em colunas, mas esta não será destacada neste artigo) contendo vários tiles. Um tile largo possui a largura de dois tiles pequenos somada à largura da margem lateral interna destes, de forma que dois tiles pequenos dispostos lateralmente ocupem exatamente a mesma área que um tile maior.

Caso seja necessário, o usuário deve poder efetuar o horizontal, ou seja, “rolar” o menu horizontalmente para acessar blocos que inicialmente não estejam visíveis na área principal da tela. Esse funcionamento se assemelha ao do menu iniciar do Windows 8, onde existe um limite vertical para disposição dos tiles. Caso um tile não caiba em uma coluna, ele deve ser posto na coluna seguinte (vale lembrar que neste artigo não será feita a divisão em colunas, essa organização se dará automaticamente ao dispor os tiles lado a lado).

Quando o usuário clicar em um dos blocos, este precisa receber algum tipo de destaque, para que fique claro que o clique funcionou. No menu que será desenvolvido aqui, os tiles terão sua cor alterada quando forem clicados.

## Aproveitando a jQuery

A jQuery é uma biblioteca JavaScript que nos fornece um conjunto vasto de funções para realizar os mais diversos tipos de operações em uma página web no lado do cliente. Um dos principais pontos fortes dessa biblioteca é a facilidade de seleção de elementos da página web a partir do uso de seletores CSS, como pela tag, id ou classe dos elementos.

Não é o objetivo desse tutorial explicar detalhadamente o funcionamento da jQuery, apenas é importante que fique claro que com ela é possível obter um ganho considerável de produtividade e, por esse motivo, nada mais natural e intuitivo que integrá-la ao projeto.

A jQuery é composta basicamente de um arquivo “js” com funções prontas e para usá-las é necessário apenas referenciar esse arquivo através de uma tag `<script>` no cabeçalho da página (o que será feito na **Listagem 1**).

O link para a página oficial da jQuery encontra-se na seção **Links**, no final do artigo.

### Observação

Na data de publicação desse artigo a última versão estável disponível era a 1.9.0, mas como foram utilizados apenas recursos simples, versões mais recentes poderão ser utilizadas normalmente.

## Mãos à obra

Para desenvolver este menu, a região central será dividida em linhas nas quais os tiles serão inseridos e organizados horizontalmente.

Os tiles serão de dois tipos: normal e largo (como se pode ver na **Figura 1**) e as cores serão dadas posteriormente com a adição de classes CSS (o leitor pode ficar à vontade para usar as cores de sua preferência).

Para começar, crie uma pasta no seu computador e reserve a mesma para guardar os arquivos do projeto. Além disso, crie também um novo arquivo de texto vazio e o nomeie como “index.html”.

Agora serão criados o título e as linhas que vão estruturar o menu, conforme mostra a **Listagem 1** a seguir.

**Listagem 1.** Estrutura inicial da página index.html

```
01 <html>
02 <head>
03   <title>Menu Metro HTML</title>
04   <link rel="stylesheet" href="estilo.css"/>
05   <meta charset="UTF-8"/>
06   <script type="text/javascript" src="http://code.jquery.com/
        jquery-1.9.0.min.js"></script>
07   <script type="text/javascript" src="script.js"></script>
08 </head>
09 <body>
10   <h1>Início</h1>
11   <div class="pagina">
12     <div class="container">
13       <div class="linha">
14       </div>
15       <div class="linha">
16       </div>
17       <div class="linha">
18       </div>
19     </div>
20   </div>
21 </body>
22 </html>
```

Note que serão utilizados dois arquivos adicionais (além do HTML onde deve ser inserido o código acima): “estilo.css” e “script.js”, além da referência à biblioteca jQuery. Portanto, crie os mesmos arquivos e salve-os junto à raiz da pasta principal do projeto.

Agora é hora de iniciar a formatação da página, para então inserir os itens do menu (tiles). No arquivo estilo.css, inicie inserindo o código constante na **Listagem 2**.

É possível perceber que no início deste código foi utilizado um recurso da CSS3: a anotação `@font-face`, que permite importar uma fonte a partir do arquivo (.ttf, .otf, etc). A fonte GOTHIC.ttf, que está sendo referenciada na mesma listagem, pode ser encontrada na pasta “C:\Windows\Fonts” no Windows 7 (e aqui foi copiada para o mesmo diretório do arquivo html), mas caso o leitor prefira, pode utilizar uma fonte nativa do seu Sistema Operacional. Para isso, bastaria remover a primeira linha da listagem e, na formatação do elemento body da regra CSS, indicar o nome da fonte nativa no lugar de Century, tal como “Arial” ou “Tahoma”, por exemplo.

Na classe CSS “linha” definimos a propriedade “display” como table, para que os tiles sejam exibidos corretamente. Isso é necessário por que utilizaremos a propriedade “float” dos tiles definida como left, com o objetivo de que eles sejam dispostos horizontalmente.

Vamos então alterar o conteúdo da div “pagina”, adicionando alguns tiles. Os tiles serão feitos com divs, pois poderemos inserir

# Metro: Criando um Menu Metro com HTML e CSS

conteúdo neles posteriormente, como imagens e texto. Para isso, utilizamos o código da **Listagem 3**.

## Listagem 2. Formatação inicial em CSS

```
01 @font-face { font-family: Century; src: url('GOTHIC.ttf');}
02
03 body{
04   font-family: Century;
05   background: #515151;
06   color: #fff;
07   padding: 20px;
08 }
09
10 .pagina{
11   width: 100%;
12   overflow-x: auto;
13   height: 100%;
14 }
15
16 .linha{
17   width: auto;
18   padding: 5px;
19   height: 110px;
20   display: table;
21 }
```

## Listagem 3. Adicionando tiles à página

```
01 <div class="container">
02   <div class="linha">
03     <div class="tile">
04     </div>
05     <div class="tile">
06     </div>
07     <div class="tile tileLargo">
08     </div>
09     <div class="tile">
10    </div>
11    <div class="tile tileLargo">
12    </div>
13  </div>
14  <div class="linha">
15    <div class="tile tileLargo">
16    </div>
17    <div class="tile">
18    </div>
19    <div class="tile">
20    </div>
21    <div class="tile">
22    </div>
23    <div class="tile tileLargo">
24    </div>
25  </div>
26  <div class="linha">
27    <div class="tile">
28    </div>
29    <div class="tile">
30    </div>
31    <div class="tile">
32    </div>
33    <div class="tile tileLargo">
34    </div>
35    <div class="tile">
36    </div>
37    <div class="tile">
38    </div>
39  </div>
40 </div>
```

Nota-se que alguns tiles possuem duas classes, “tile” e “tileLargo”. A classe “tile” é genérica e servirá para definir algumas propriedades comuns aos dois tipos de item, como a altura e a margem. A classe “tileLargo”, por sua vez, terá apenas a largura aumentada em relação a outra.

Vejamos então a **Listagem 4** onde são definidas as formatações iniciais dos tiles.

## Listagem 4. Formatação inicial dos tiles

```
01 .tile{
02   height: 100px;
03   width: 100px;
04   float: left;
05   margin: 0 5px 0 0;
06   padding: 2px;
07 }
08
09 .tileLargo{
10   width: 210px;
11 }
```

A propriedade “float” foi definida com o valor `left` para que os tiles de uma mesma linha sejam dispostos lateralmente. As dimensões utilizadas foram `100x100px` por padrão, mas o leitor pode alterar para os valores de sua preferência e que mais se adequem a sua necessidade.

É importante que a classe `tileLargo` seja definida após a classe `tile`, para que ao carregar a página, a largura de `210px` seja aplicada após a largura inicial de `100px`, sobrepondo-a. Caso contrário, a configuração que prevaleceria seria a da classe `tile`.

### Observação

A largura de `210px` foi calculada como sendo o dobro da largura de um tile básico (pequeno), somado às duas margens laterais de `5px` cada.

Ao abrir o arquivo no browser ainda não será possível ver o menu como gostaríamos. Isso ocorre apenas porque a cor dos tiles não foi definida. Portanto, criaremos algumas classes no arquivo CSS contendo apenas a cor do `background`. Neste exemplo, foram utilizadas quatro cores, como pode ser visto a seguir, na **Listagem 5**.

Da mesma forma, estas cores e classes também podem ser customizadas segundo a preferência do leitor, apenas seguindo a lógica sugerida. Caso deseje, por exemplo, pode usar uma imagem de plano de fundo, alterar a cor da borda e da fonte dos blocos.

O próximo passo é associar os tiles às classes de cores. No exemplo deste artigo, o conteúdo da div “pagina” foi alterado conforme a **Listagem 6**, a seguir, definindo as classes dos blocos aleatoriamente.

Agora sim é possível visualizar o resultado, que deve ser semelhante à **Figura 2**.

Os códigos apresentados até aqui são consideravelmente simples e de fácil entendimento. Talvez uma segunda leitura sobre os

mesmos seja necessária para assimilar melhor sua estrutura, e isso é algo diretamente relacionado ao seu nível de conhecimento em HTML e CSS, então tenha calma se não entender de imediato.

Vamos apenas aplicar algumas configurações adicionais, colocando um texto e imagem dentro de cada tile e deixando-os mais próximos do exemplo real. A **Listagem 7** mostra como ficou um tile após estas mesmas alterações. Logo, com base nele os demais podem ser feitos igualmente.

#### Listagem 5. Classes de cores dos tiles

```
01 .amarelo{  
02   background:#DAA520;  
03 }  
04  
05 .vermelho{  
06   background:#CD0000;  
07 }  
08  
09 .azul{  
10  background:#4682B4;  
11 }  
12  
13 .verde{  
14  background-color: #2E8B57;  
15 }
```

#### Listagem 6. Associação dos tiles às cores

```
01 <div class="pagina">  
02   <div class="linha">  
03     <div class="tile amarelo">  
04     </div>  
05     <div class="tile azul">  
06     </div>  
07     <div class="tileLargo vermelho">  
08     </div>  
09     <div class="tile verde">  
10    </div>  
11     <div class="tile tileLargo amarelo">  
12     </div>  
13   </div>  
14   <div class="linha">  
15     <div class="tileLargo amarelo">  
16     </div>  
17     <div class="tile azul">  
18     </div>  
19     <div class="tile verde">  
20     </div>  
21     <div class="tile vermelho">  
22     </div>  
23     <div class="tileLargo verde">  
24     </div>  
25   </div>  
26   <div class="linha">  
27     <div class="tile amarelo">  
28     </div>  
29     <div class="tile verde">  
30     </div>  
31     <div class="tile vermelho">  
32     </div>  
33     <div class="tileLargo verde">  
34     </div>  
35     <div class="tile azul">  
36     </div>  
37     <div class="tile verde">  
38     </div>  
39   </div>  
40 </div>
```

Como inserimos novos elementos nos blocos, é preciso formatá-los para que sejam exibidos corretamente. No exemplo da **Listagem 8** foi definido que as imagens terão as dimensões 56x56px e, com base nisso, foi calculado o valor das suas bordas laterais e superior, de forma que fiquem centralizadas nos blocos.

Após inserir algumas imagens, seguindo o modelo da **Listagem 7**, temos agora o seguinte conteúdo no arquivo HTML da **Listagem 9** e um novo resultado, que é exibido na **Figura 3**.

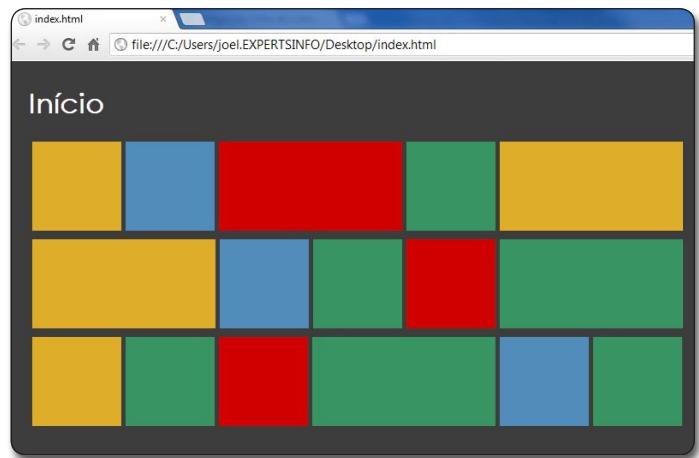


Figura 2. Resultado dos tiles com cores definidas

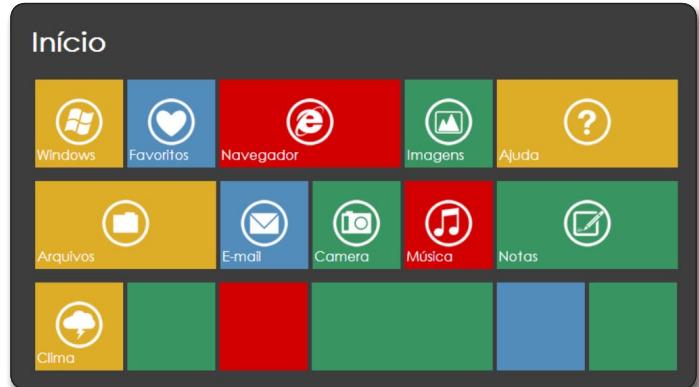


Figura 3. Novo layout da página com imagens nos blocos

#### Listagem 7. Exemplo de tile com título

```
<div class="tile amarelo">  
    
  <span>Windows</span>  
</div>
```

#### Listagem 8. Configuração das imagens dentro dos blocos

```
.tile img{  
  width:56px;  
  margin:20px 22px 0 22px  
}  
  
.tileLargo img {  
  margin:20px 77px 0 77px  
}
```

# Metro: Criando um Menu Metro com HTML e CSS

**Listagem 9.** Código da página com imagens nos tiles

```
01<div class="pagina">
02 <div class="container">
03   <div class="linha">
04     <div class="tile amarelo">
05       
06       <span>Windows</span>
07     </div>
08     <div class="tile azul">
09       
10      <span>Favoritos</span>
11    </div>
12    <div class="tile tileLargo vermelho">
13      
14      <span>Navegador</span>
15    </div>
16    <div class="tile verde">
17      
18      <span>Imagens</span>
19    </div>
20    <div class="tile tileLargo amarelo">
21      
22      <span>Ajuda</span>
23    </div>
24  </div>
25  <div class="linha">
26    <div class="tile tileLargo amarelo">
27      
28      <span>Arquivos</span>
29    </div>
30    <div class="tile azul">
31      
32      <span>E-mail</span>
33   </div>
34   <div class="tile verde">
35     
36     <span>Camera</span>
37   </div>
38   <div class="tile vermelho">
39     
40     <span>Música</span>
41   </div>
42   <div class="tile tileLargo verde">
43     
44     <span>Notas</span>
45   </div>
46 </div>
47 <div class="linha">
48   <div class="tile amarelo">
49     
50     <span>Clima</span>
51   </div>
52   <div class="tile verde">
53   </div>
54   <div class="tile vermelho">
55   </div>
56   <div class="tile tileLargo verde">
57   </div>
58   <div class="tile azul">
59   </div>
60   <div class="tile verde">
61   </div>
62 </div>
63 </div>
64 </div>
```

Para facilitar a comparação entre o antes e o depois, essa figura também exibe a representação de alguns tiles sem imagem.

Até este momento, como se pode ver na figura anterior, foi inserida uma pequena quantidade de blocos, de forma que o conjunto não toma sequer a largura total da página. Porém, sabemos que em um site real podem existir várias opções de menu, sendo necessário criar tanto novas linhas quanto colunas com mais blocos.

Nessa situação, é preciso que seja possível efetuar o *scroll* horizontal, ou seja, o usuário deve poder “rolar” o menu horizontalmente (seguindo o modelo do Windows 8, expande-se o menu apenas lateralmente e não na direção vertical).

Para suprir essa necessidade faremos uso de JavaScript, mais especificamente da biblioteca jQuery, que facilita a manipulação dos elementos da página e nos permite tornar a interface mais dinâmica e agradável aos olhos do usuário.

Observando a **Listagem 2** é possível ver as seguintes configurações:

- A propriedade *width* da div “pagina” foi definida com o valor 100%, para que ocupe toda a largura do corpo da página, não mais que isso. Caso a largura da div aumentasse dinamicamente, não

veríamos a barra de rolagem como desejado, a barra exibida seria a da página como um todo devido à div ter excedido os limites horizontais.

A propriedade *overflow-x* da mesma div foi definida como *auto*, o que indica que a barra de rolagem horizontal será exibida quando a largura do conteúdo da div for maior que a sua própria largura. Nesse caso, isso ocorrerá quando a largura da div “container” superar a largura da div “pagina” (por isso temos a div container).

Tendo entendido o porquê da atribuição de tais valores às propriedades citadas, podemos usar essas informações para exibir a barra de rolagem na div “pagina”. O que precisamos é que a div container tenha sua largura aumentada dinamicamente de acordo com a quantidade de blocos existentes.

Para isso, usaremos um script que percorrerá todas as linhas de tiles (divs pertencentes à classe linha) e calculará a largura total dos blocos de cada uma, considerando inclusive os valores das margens laterais. Como são várias linhas e elas não necessitam ter a mesma quantidade de blocos, será verificado qual linha possui maior largura total, então esse valor será atribuído à propriedade

*width* da div “container”. Isso fará com que a largura da div interna (container) exceda os limites da externa (página), fazendo com que a barra de rolagem seja apresentada para o usuário. Então, no arquivo script.js (que até então deve estar vazio) adicione o conteúdo da **Listagem 10**.

#### **Listagem 10.** Arquivo script.js com função para apresentar a barra de rolagem

```
01 $(function () {
02   var maiorLinha = 0;
03   $(".container .linha").each(function () {
04     var larguraLinha = 0;
05     $(this).children(".tile").each(function () {
06       larguraLinha += $(this).width();
07       larguraLinha += 2 * parseInt($(this).css("margin-right").toString());
08       replace("px", "");
09     });
10     if (larguraLinha > maiorLinha)
11       maiorLinha = larguraLinha + 5;
12   });
13 });

12 $(".container").css("width", maiorLinha.toString() + "px");
13});
```

Para percorrer as linhas da página e os blocos de cada linha, foi utilizada a função “*each()*”, que permite iterar sobre os itens de uma lista. Na mesma listagem foi usada em dois momentos a palavra reservada *this*, nos quais possui significados diferentes. Na primeira ocorrência o *this* refere-se à própria div “*linha*”, da qual filtramos os elementos filhos (*children*) que pertencem à classe tile e iteramos sobre estes elementos. No outro uso do identificador *this*, este já se refere ao bloco que está sendo acessado em cada iteração, pois está sendo usado dentro da segunda função *each*.

#### Observação

Entenda-se por iterar percorrer todos os elementos de uma lista com o objetivo de efetuar determinada ação sobre os mesmos.

Após calcular a largura, atribuímos esse valor à propriedade *width* da div container usando a função *css()* da jQuery, função essa que recebe dois parâmetros referentes ao nome da propriedade CSS e o valor a ser aplicado respectivamente. Como a propriedade *width* se trata de um atributo CSS, é preciso informar corretamente a sintaxe, informando a unidade dessa medida, no caso, pixels (px).

Agora podemos inserir mais tiles horizontalmente e observar o resultado na **Figura 4**.

Para finalizar, resta aplicar um pequeno efeito aos tiles para que eles tenham a aparência alterada ao serem clicados. Para isso usaremos as seguintes funções da jQuery (tal como pode ser visto na **Listagem 11**):

- **mousedown**, trata o evento de pressionamento do botão do mouse sobre o elemento HTML;

- **mouseup**, trata o evento de soltura do pressionamento do botão do mouse sobre o elemento HTML;
- e **mouseleave**, trata o evento de remoção do cursor do mouse de sobre o elemento HTML.



**Figura 4.** Menu com barra de rolagem horizontal

#### Nota

Como até aqui já foram apresentados vários exemplos de uso dos tiles, não será mostrado aqui o novo conteúdo da página HTML com os novos blocos adicionados. O leitor pode inseri-los segundo sua preferência de quantidade e cor, desde que somem uma largura suficiente para que se observe a barra de rolagem.

#### **Listagem 11.** Tratamento dos eventos de botão do mouse sobre os tiles

```
01 $(".tile").mousedown(function () {
02   $(this).addClass("selecionado");
03 });
04
05 $(".tile").mouseup(function () {
06   $(this).removeClass("selecionado");
07 });
08 $(".tile").mouseleave(function () {
09   $(this).removeClass("selecionado");
10 });
```

Quando o botão do mouse for pressionado, será adicionada a classe CSS “selecionado” ao tile e quando o botão for solto esta classe será removida. O uso da função *mouseleave* foi necessário para corrigir uma pequena falha que surge se usarmos apenas as duas primeiras. Se o usuário pressionar o botão do mouse e, ainda com o botão pressionado, remover o cursor sobre o tile, este não volta à sua configuração original, pois o evento *mouseup* não é disparado. Por isso fazemos com que o bloco volte ao normal se o cursor estiver sobre ele e for removido.

A classe “selecionado” (ver **Listagem 12**) apenas altera a cor de fundo do tile, mas outros efeitos poderiam ser aplicados apenas adicionando mais código à listagem (também a critério do leitor).

# Metro: Criando um Menu Metro com HTML e CSS

Dessa forma o menu está pronto para uso. Podemos visualizar o resultado final na **Figura 5**, onde o primeiro bloco aparece pressionado e, consequentemente, com a cor de fundo mudada.

Com isso finalizamos nossa página. Não é a mesma coisa que o menu do Windows 8, mas é, sem dúvidas, uma solução viável para esta situação, com esse tipo de design.



Figura 5. Menu funcionando com um tile pressionado

Possuindo algum conhecimento de HTML, CSS e JavaScript, o leitor poderá alterar facilmente a aparência do menu e atribuir funcionalidades aos tiles como, por exemplo, usá-los como links ou para invocar funções JavaScript.

Espero que o conteúdo deste artigo possa ser útil para os profissionais do desenvolvimento web interessados em desenvolver um layout no estilo metro, uma das novas tendências no mercado de software da atualidade.

## Autor



### Joel Rodrigues

[joelrlneto@gmail.com](mailto:joelrlneto@gmail.com)



Técnico em Informática, formado pelo Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, graduando em Ciências e Tecnologia na Universidade Federal do Rio Grande do Norte, atualmente o time de desenvolvimento da Expert's Informática, onde participa de projetos para diversos segmentos. Atua há mais de três anos no desenvolvimento de sistemas com Delphi e .NET, além das Web Standards. É, ainda, autor de mais de uma centena de artigos nos portais DevMedia e Linha de Código.

### Listagem 12. Classe para destacar o tile clicado

```
.selecionado{  
background-color: #483D8B;  
}
```

## Links:

[Página oficial da biblioteca jQuery](http://www.jquery.com)

[www.jquery.com](http://www.jquery.com)

RENOVE JÁ!

Sua assinatura pode estar acabando

Renovando a assinatura de sua revista favorita  
você ganha brindes e descontos exclusivos.

# Somos tão apaixonados por tecnologia que o nome da empresa diz tudo.

**Porta 80** é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.  
Somos uma equipe composta de gente que entende e gosta do que faz,  
**assim como você.**



## Estrutura

100% NACIONAL.  
Servidores de primeira linha, links de alta capacidade.



## Supporte diferenciado

Treinamos nossa equipe para fazer mais e melhor.  
Muito além do esperado.



## Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.



## 1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única.  
Conheça!



# Porta 80

WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce |  
Aplicações | Streaming | Email corporativo

[porta80.com.br](http://porta80.com.br) | [comercial@porta80.com.br](mailto:comercial@porta80.com.br) | [twitter.com/porta80](http://twitter.com/porta80)

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486