

Front-end magazine

Edição 13

 DEVMEDIA

Game em Pixi.js
O poder do JavaScript na
construção de jogos online

Angular Material
Design flexível e responsivo na
web com AngularJS
e o Google Material Design

RAILS + BOOTSTRAP

**Aplicações responsivas
na web com Ruby on Rails**



MVP

R\$ 1.000.000,00
INVESTIDOS EM CONTEÚDO
NOS ÚLTIMOS 12 MESES.

APLIQUE ESSE INVESTIMENTO
NA SUA CARREIRA...

E MOSTRE AO MERCADO
QUANTO VOCÊ VALE!

CONFIRA TODO O MATERIAL
QUE VOCÊ TERÁ ACESSO:

- + de **9.000** video-aulas
- + de **290** cursos online
- + de **13.000** artigos
- DEVMEDIA API's consumido + de **500.000** vezes

POR APENAS
R\$ 69,90* mensais

*Tempo mínimo de assinatura: 12 meses.



PRA QUEM QUER EXIGIR
MAIS DO MERCADO!



 **DEVMEDIA**

EXPEDIENTE

Editor

Diogo Souza (diogosouzac@gmail.com)

Consultora Técnica

Anny Caroline (annycarolinegnr@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, ou se preferir entre em contato conosco através do telefone 21 3382-5038.

Publicidade

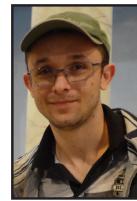
publicidade@devmedia.com.br – 21 3382-5038

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

diogosouzac@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.

Sumário

Conteúdo sobre Novidades

04 – Primeiros passos com o Angular Material

[*Fabrício Galdino*]

Destaque-Mentoring

16 – Rails + Bootstrap: Integrando as tecnologias na criação de aplicações web

[*Fabrício Galdino*]

Artigo no estilo Curso

30 – Desenvolvimento de jogos web com Pixi.js – Parte 3

[*Júlio Sampaio*]

Primeiros passos com o Angular Material

Aprenda a unir o poder do Material Design com a flexibilidade do AngularJS e suas diretivas dinâmicas

Na última conferência do Google I/O em 2014, o Google anunciou o Material Design, a sua nova linguagem de design, tanto para a web quanto para dispositivos móveis. Eles já haviam convertido grande parte de suas aplicações populares para aderir a essa nova especificação, se esforçando para fornecer uma experiência consistente a seus usuários.

Depois de conhecer o Material Design, você irá obter imediatamente um sentimento minimalista ultramoderno em relação a suas aplicações. O objetivo é criar uma linguagem visual que sintetize princípios clássicos de um bom design juntamente à inovação e tecnologia. Também será possível desenvolver um único sistema subjacente que permite uma experiência unificada em várias plataformas e em diversos tamanhos de dispositivos. O Material Design é fundado em três princípios fundamentais:

- **O Material é a metáfora:** inspirado pelo estudo do papel e da tinta, o material vive no espaço 3D e está fundamentado na realidade tátil (ele dá a ilusão de espaço usando sombras realistas). O material de papel deve respeitar as leis da física (ou seja, dois pedaços de papel não podem atravessar um ao outro), mas pode suplantar o mundo físico (isto é, um papel pode aumentar ou diminuir);

- **Negrito, gráfico e intencional:** o framework trabalha com o conceito de mistura de cores, com alta definição e relevo, além do uso constante de imagens de vários tipos e definições, bem como a presença de uma tipografia escalada e espaços em branco destinados a criar uma interface ousada e gráficos que mergulham o usuário na experiência. O botão de ação flutuante, ou *FAB* (*Float Action Button*), é um excelente exemplo desse princípio. Você já reparou alguma vez no pequeno círculo com o símbolo “+” flutuando em seu aplicativo Google na caixa de entrada? O Material Design torna muito evidente que esse é um botão importante;

Fique por dentro

Desenvolver aplicações web sempre requer um enorme esforço de design, estruturação e modularização dos templates que serão reaproveitados ao longo do ciclo de desenvolvimento da solução, principalmente nos dias de hoje, onde os requisitos de aparência de um site são cada vez mais exigentes. Em função disso, esse artigo se mostra útil ao abordar tal realidade através do framework Angular Material. Trata-se de uma união mais que bem-vinda dos já idealizados frameworks do Google: Material Design (especificação oficial da empresa para designs na web/mobile) e o AngularJS. Aqui, você aprenderá a usufruir ao máximo dos recursos de ambos através da construção de exemplos simples e funcionais.

- **O movimento fornece um significado:** o movimento é algo importante no design atual porque atrai o usuário e o prende ao efeito que está ocorrendo no espaço da tela. Além disso, a sutileza passa também a impressão de profissionalismo no que se refere ao design como um todo, trazendo mais confiança para o usuário. O ponto principal aqui é fazer uso de animações somente quando se tem um propósito de fato e sem exageros.

Como o AngularJS cabe no Material Design?

O AngularJS, o framework JavaScript MVW (um acrônimo alternativo ao MVC, que significa *Model-View-Whatever*) do Google, aborda muitos dos desafios encontrados no desenvolvimento de aplicações *single-page*. Ele fornece o framework necessário para a criação de aplicações web modernas que se conectam a APIs e nunca precisam de uma página para serem atualizadas.

O Angular seria a HTML se tivesse sido projetada para aplicações. A HTML é uma grande linguagem declarativa para documentos estáticos, porém nem tanto para a criação de aplicações dinâmicas. A criação de aplicações dinâmicas com a HTML tem sido sempre um exercício de enganar o navegador para fazer coisas que não deveria fazer. Existe um par de abordagens para fazer isso:

- 1. Biblioteca** – uma coleção de funções. (Por exemplo: o jQuery)
- 2. Framework** – o código preenche dinamicamente os elementos estáticos, quando necessário.

O Angular tem uma abordagem diferente para resolver esse problema. Ao invés de lutar com o HTML dado, ele cria novas construções HTML. O Angular ensina ao navegador uma nova sintaxe HTML através de uma construção chamada “diretivas”. Além de vir com um conjunto delas internamente, também permite que criemos diretivas personalizadas, podendo escrever nossos próprios elementos HTML.

Introdução ao Angular Material

O Google está desenvolvendo ativamente o Angular Material, uma implementação do Material Design no AngularJS. O Angular Material é composto por várias peças, possui uma biblioteca de CSS para a tipografia e outros elementos, fornece uma abordagem JavaScript interessante para *theming* (definição de temas) e seu layout responsivo usa uma grade do tipo *flex*. Mas a característica mais atraente do Angular Material é a sua incrível coleção de diretivas. Existem duas maneiras básicas que podemos seguir para usar o Angular Material:

- Instalação local – podemos baixar as bibliotecas utilizando o *npm*, *jspm* ou *bower* em sua máquina local, incluindo-as em seu código HTML.
- Versão baseada no CDN – podemos incluir os arquivos *angular-material.min.css* e *angular-material.js* em seu código HTML diretamente via *Content Delivery Network* (CDN), isto é, via URL remota disponibilizada pelo site da própria fabricante.

Neste artigo trataremos de explorar alguns dos principais recursos do framework do Angular Material, explorando desde a sua instalação até a importação dos arquivos nas páginas HTML em conjunto com os recursos nativos do AngularJS. O leitor terá, ao final do mesmo, insumos suficientes para entender como ambos os frameworks se comunicam, bem como adaptá-los aos seus projetos facilmente. Além disso, trataremos de expor a integração entre a estrutura HTML com os códigos de estilo do CSS que a biblioteca do Angular Material disponibiliza por padrão para incutir o design característico do framework e seus componentes.

Instalação local

Para executar os comandos desse artigo, você precisará ter o Node.js instalado na máquina, assim poderemos fazer uso do seu gerenciador de pacotes, o *npm*. Use o seguinte comando *npm* para instalar as bibliotecas:

```
npm install angular-material
```

O *npm* irá baixar os arquivos do Angular Material no diretório *node_modules > angular-material*. Por isso, é importante selecionar um diretório para guardar seus arquivos no projeto final. Vejamos o exemplo demonstrado na **Listagem 1**, um Alô Mundo usando

os módulos iniciais do Angular Material. Veja que ele traz apenas uma série de importações dos arquivos de CSS e JavaScript que o framework faz uso. Como de praxe, toda aplicação AngularJS deve conter no *body* da página o *ng-app* com o nome do módulo da aplicação. Esse será usado no JavaScript que inicializa o módulo do *ngMaterial*.

Obteremos o resultado produzido conforme mostra a **Figura 1**.

Versão baseada no CDN

Você também pode implementar o mesmo exemplo usando os arquivos *angular-material.min.css* e *angular-material.min.js* do Google CDN, como mostra a **Listagem 2**. O resultado, por sua vez, continuará o mesmo.

Alô Mundo Angular Material

Lore ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lore ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figura 1. Resultado exibido do Alô Mundo

Listagem 1. Alô Mundo Angular Material.

```
<html lang="pt">
<head>
<meta charset="utf-8" />
<link rel="stylesheet" href="node_modules/angular-material/angular-material.css"/>
</head>
<body ng-app="aloMundo" ng-cloak>
<md-toolbar class="md-warn">
<div class="md-toolbar-tools">
<h2 class="md-flex">Alô Mundo Angular Material</h2>
</div>
</md-toolbar>
<md-content flex layout-padding>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</p>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</p>
<p>Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
</md-content>
</body>
<script src="node_modules/angular/angular.js"></script>
<script src="node_modules/angular-animate/angular-animate.js"></script>
<script src="node_modules/angular-aria/angular-aria.js"></script>
<script src="node_modules/angular-messages/angular-messages.js"></script>
<script src="node_modules/angular-material/angular-material.js"></script>
<script type="text/javascript">
angular.module('aloMundo', ['ngMaterial']);
</script>
</html>
```

Primeiros passos com o Angular Material

Listagem 2. Instalando o Angular Material CDN

```
<html lang="pt">
<head>
<meta charset="utf-8"/>
<link rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/
angular_material/1.0.0/angular-material.min.css">
</head>
<body ng-app="aloMundo" ng-cloak>
<md-toolbar class="md-warn">
<div class="md-toolbar-tools">
<h2 class="md-flex">Alô Mundo Angular Material</h2>
</div>
</md-toolbar>
<md-content flex layout-padding>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. </p>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. </p>
<p>Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore
eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
</md-content>
</body>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
angular.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
angular-animate.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
angular-aria.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
angular-messages.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angular_material/1.0.0/
angular-material.min.js"></script>
<script type="text/javascript">
angular.module('aloMundo', ['ngMaterial']);
</script>
</html>
```

Diretivas

As diretivas são uma característica do núcleo do AngularJS. Como já falamos anteriormente, o Angular vem com várias diretivas que usamos todo o tempo, como *ng-model* ou *ng-repeat*. Elas são uma parte muito importante do framework, pois disponibilizam ações customizadas e reaproveitamento de código.

Essa biblioteca possui um conjunto de diretivas que inspiram o design Material. Diretivas do Angular Material são tags HTML que começam com *md* (abreviação de material design), e são bem fáceis de usar. Por exemplo, vamos dar uma olhada em um exemplo de botão simples. Um botão HTML padrão poderia ser algo como:

```
<button>Clique aqui!</button>
```

Um botão do Angular Material se parece mais ou menos com isso:

```
<md-button>Clique aqui!</md-button>
```

E isso é tudo o que precisamos para fazer um botão Material. Agora, existem várias outras opções que estão disponíveis para essa diretiva, como *theming* e elevação a partir da superfície, para aumentar a sua importância:

```
<md-button class="md-raised md-primary md-hue-1">Clique aqui!</md-button>
```

A classe *md-raised* se encarrega de incutir o efeito elevado ao botão, com uma sombra no plano de fundo, tal como vemos na **Figura 2**.



Figura 2. Exemplo do botão com classe md-raised

Serviços

Os serviços são de igual modo fundamentais para a funcionalidade do Angular, utilizados para dividir o código através da aplicação. Um serviço comum de núcleo como o *\$http* é usado e reutilizado para chamadas de dados em aplicações do Angular. Os serviços Angular são:

- *Lazily instantiated* (preguiçosamente instanciados) – O Angular apenas instancia um serviço quando um componente da aplicação depende dele;
- *Singletons* – cada componente dependente de um serviço obtém uma referência à única instância gerada pela fábrica de serviços.

O Angular Material vem embalado com alguns serviços que fornecem funcionalidades extras para nossas aplicações. Eles também contribuem para o desempenho de algumas das diretivas. Um grande exemplo de um desses serviços é o *toast*. Um *toast* é uma pequena notificação que desliza a partir da parte inferior da tela e desaparece após alguns segundos. Veja na **Listagem 3** como implementar esse tipo de serviço no JavaScript. Para o recurso funcionar, precisamos incorporar o módulo do *\$mdToast* ao controller do AngularJS, assim ele estará disponível para instanciar a função *show()* e exibir a mensagem. Para o exemplo funcionar, você precisa adicionar também um *ng-controller* à tag body da página, uma vez que ele será mapeado pela função *controller()* na declaração do módulo do Angular:

```
<body ng-app="aloMundo" ng-controller="ToastEx" ng-cloak>
```

O resultado pode ser visualizado na **Figura 3**. Este exemplo mostra apenas um *toast* simples que aparece no canto inferior esquerdo da tela e é removido após três segundos.

Theming

O Material Design é uma linguagem visual onde os temas transmitem significados através de cores, tons e contraste.

Listagem 3. Criando um toast simples.

```
angular.module('aloMundo', ['ngMaterial']).controller('ToastEx', function($scope, $mdToast) {
  $mdToast.show(
    $mdToast.simple('Alô Mundo Toast!')
      .position('left bottom')
      .hideDelay(3000)
  );
});
```

Alô Mundo Angular Material

Consectetur adipiscing

elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Consectetur adipiscing

elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Alô Mundo Toast!

Figura 3. Mensagem sendo exibida num Toast

Esses temas são expressos ao longo dos componentes do aplicativo para proporcionar uma sensação mais unificada.

De acordo com as diretrizes do Material Design, você deve limitar a seleção de cores, escolhendo três tonalidades de cores da paleta primária e uma cor de destaque da paleta secundária. Ele segue essa simples diretriz usando o JavaScript para configurar o tema. Mas qual a diferença entre uma paleta e uma tonalidade? A tonalidade é uma única cor em uma paleta, já a paleta é um conjunto de tonalidades. Por exemplo uma paleta seria verde e uma tonalidade é um determinado tom de verde.

Configurando o tema

Tematizar o seu projeto é uma opção muito útil para impor um design profissional sem grandes esforços no Angular Material. No arquivo app.js (crie-o no diretório /js para organizar melhor a distribuição dos arquivos do projeto), é preciso definir suas paletas e tonalidades usando o serviço de provedor de *theming*:

conforme podemos ver na **Listagem 4**. Nela, temos algumas funções importantes, a saber:

- A função config() define a função que lidará com as definições do tema a ser usado pelo Angular no design do módulo *myApp*. Veja que ela recebe o parâmetro *mdThemingProvider*, o qual se responsabiliza por fornecer os métodos da API do Material para customizar todos os componentes do framework;
 - A função theme() auxilia nesse processo como um todo recebendo um parâmetro com o tipo do tema a ser usado (valor “default” configura o padrão);
 - Já a função primaryPalette() define o início das nossas customizações: a cor primária será o azul, com matizes distintas definidas através dos elementos de prefixo *hue*, os quais se encarregam de aumentar/diminuir ligeiramente a tonalidade da cor em questão;
 - Por último, as funções accentPalette(), warnPalette() e backgroundPalette() recebem as strings referentes às cores que configuraram a paleta, mensagens de alerta e plano de fundo, respectivamente.

Listagem 4. Definindo paletas e tonalidade em um theming

```
angular.module('myApp', ['ngMaterial'])
.config(function($mdThemingProvider) {
  $mdThemingProvider.theme('default')
    .primaryPalette('blue', {
      'default': '400',
      'hue-1': '100',
      'hue-2': '600',
      'hue-3': 'A100'
    })
    .accentPalette('orange')
    .warnPalette('yellow')
    .backgroundPalette('grey');
});
```

Na **Listagem 5** aplicamos o tema nos componentes definindo a classe do elemento para a paleta e tonalidade desejada. Veja que precisamos referenciar cada uma das paletas criadas exatamente pelo seu nome (warn, accent, etc.). Veja o resultado de tal ação na **Figura 4**.

[CLIQUE AQUI!!](#) [CLIQUE AQUI!!](#) [CLIQUE AQUI!!](#) [OUTA VEZ AQUI](#) [CUIDADO](#)

Figura 4. Resultado dos botões com as classes da paleta

Listagem 5. Exemplos de botões Material com tema que criamos.

```
<md-button class="md-primary">Clique aqui!</md-button>
<md-button class="md-primary md-hue-1">Clique aqui!</md-button>
<md-button class="md-primary md-hue-2">Clique aqui!</md-button>
<md-button class="md-accent">ou talvez aqui!</md-button>
<md-button class="md-warn">Cuidado!</md-button>
```

Primeiros passos com o Angular Material

Layouts

O *flexbox* é a maior e mais recente adição de design ao Angular Material. Trata-se do sistema nativo do CSS3 que lida com a organização automática dos elementos de uma página quando exibida em dispositivos de diferentes tamanhos, dispensando a propriedade *float*, antes largamente usada como solução para impor responsividade. Se você estiver familiarizado com o sistema de grids do Bootstrap, terá mais facilidade ao utilizá-lo, pois na verdade o Bootstrap já está migrando para o *flexbox* em seu próximo lançamento. Ele é baseado no sistema de layout de linhas e colunas (geralmente em máximo de 12), porém com muito mais recursos.

A diretiva de layout é um elemento de container usado para especificar a direção do layout para os elementos filhos. A seguir estão os valores atribuíveis à mesma:

- **row** – os itens são dispostos horizontalmente, com *max-height* = 100% e *max-width* igual à largura dos itens no recipiente;
- **column** – os itens são dispostos verticalmente, com *max-width* = 100% e *max-height* igual à altura dos itens no recipiente.

De acordo com as diretivas do design responsivo, com a forma como o layout será alterado e dependendo do tamanho da tela do dispositivo, as seguintes definições de layout definidas na **Tabela 1** podem ser usadas para definir a direção nos dispositivos com larguras de diferentes dimensões.

| Layout | Dimensões |
|--------------|--------------------------|
| layout-xs | width < 600px |
| layout-gt-xs | width >= 600px |
| layout-sm | 600px <= width < 960px |
| layout-gt-sm | width >= 960px |
| layout-md | 960px <= width < 1280px |
| layout-gt-md | width >= 1280px |
| layout-lg | 1280px <= width < 1920px |
| layout-gt-lg | width >= 1920px |
| layout-xl | width >= 1920px |

Tabela 1. Lista de dimensões para diferentes layouts (Fonte: documentação oficial do Material Angular)

A **Listagem 6**, por sua vez, mostra como usar essa diretiva para exibir o layout padrão.

Veja na **Figura 5** o resultado. Observe que na listagem estamos fazendo uso dos atributos *layout* definidos nas tags `<div>` para determinar se as mesmas adotarão o estilo *row* ou *column*, conforme mencionamos. A função `layoutController()` está vazia já que não configuraremos nada para o controller ainda e o Angular não permite inicializar um controle sem uma função associada. Veja como o Angular Material trabalha o resultado final em detrimento das concepções de agrupamento que configurarmos.

A diretiva *flex* em um elemento de container é usada para personalizar o tamanho e a posição dos elementos, ela define a



Figura 5. Diretiva de layout padrão

Listagem 6. Exemplo de uso da diretiva com o layout padrão.

```
<html lang="pt">
<head>
  <meta charset="utf-8" />
  <link rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/
    angular_material/1.0.0/angular-material.min.css">
  <style>
    .box {
      color:white;
      padding:10px;
      text-align:center;
      border-style: inset;
    }
    .red {
      background:red;
    }
    .orange {
      background:orange;
    }
  </style>
</head>

<body ng-app="firstApplication">
<div id="layoutContainer" ng-controller="layoutController as ctrl"
  style="height:100px;" ng-cloak>
  <div layout="row" layout-xs="column">
    <div flex class="red box">Linha 1: Item 1</div>
    <div flex="20" class="orange box">Linha 1: Item 2</div>
  </div>
  <div layout="column" layout-xs="column">
    <div flex="33" class="red box">Coluna 1: item 1</div>
    <div flex="66" class="orange box">Coluna 1: item 2</div>
  </div>
</div>
</body>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular-animate.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular-aria.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular-messages.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angular_material/1.0.0/
  angular-material.min.js"></script>
<script type="text/javascript">
  angular.module('firstApplication', ['ngMaterial'])
    .controller('layoutController', layoutController);

  function layoutController ($scope) {
  }
</script>

</html>
```

maneira como o elemento deve ajustar o seu tamanho em relação ao seu recipiente pai e aos outros elementos dentro do container. A seguir estão os valores atribuíveis:

- **Múltiplos de 5.** Ex: 5, 10, 15 ... 100
- **33** – equivalente a 33%
- **66** – equivalente a 66%

A **Listagem 7** mostra o uso dessa diretiva demonstrando como atribuir o *flex* às nossas páginas.

Veja na **Figura 6** o resultado. Observe que as definições se mantiveram em relação à configuração do controller e das diretivas padrão do Angular, porém apenas acrescentamos o atributo *flex* aos nossos elementos de `<div>` definindo seus valores divididos tal como demonstramos. Bem simples, não acha?

A seguir vamos conhecer algumas das melhores diretivas do Angular Material, onde teremos alguns exemplos práticos de como aplicá-las em nosso código.



Figura 6. Usando a diretiva de layout flex

Autocomplete

O *Autocomplete* fornece uma experiência agradável ao usuário, o auxiliando na escolha de uma opção. É o que faz o motor de busca do Google, o melhor do mercado. A diretiva *Autocomplete* adiciona essa funcionalidade ao seu aplicativo através da exibição do preenchimento das palavras via sugestões à medida em que o usuário as escreve. Mas a melhor parte dessa diretiva é a personalização. Ao preencher o seu *autocomplete* com o atributo *md-item-template* você pode dar mais sentido às sugestões. Por exemplo, se um usuário estava à procura de nomes em uma empresa, o *autocomplete* poderia mostrar os nomes correspondentes com a imagem da sua empresa (assim como vemos ao buscar alguém no Facebook ou no LinkedIn), dando ao usuário uma experiência mais robusta. A tag `<md-autocomplete>` pode ser usada para fornecer resultados de pesquisa a partir de dados locais, remotos ou em cache através do código *sources.md-autocomplete*. Após a primeira chamada, ele usa os resultados em cache para eliminar solicitações desnecessárias do servidor bem como lógicas de pesquisa que podem ser desativadas.

Vejamos uma série de atributos e suas descrições que usaremos no nosso exemplo de autocomplete:

- **md-no-cache:** desativar o cache interno que acontece no *autocomplete*;
- **md-items:** uma expressão que recebe a lista de itens a serem iterados no componente de pesquisa;
- **md-selected-item-change:** uma expressão a ser executada cada vez que um novo item é selecionado;
- **md-search-text-change:** uma expressão a ser executada ao atualizar os textos da pesquisa;

Listagem 7. Atribuindo atributo flex à nossa página.

```
<html lang="pt">
<head>
<meta charset="utf-8" />
<link rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/
angular_material/1.0.0/angular-material.min.css">
<style>
.box {
  color:white;
  padding:10px;
  text-align:center;
  border-style: inset;
}
.red {
  background:red;
}
.orange {
  background:orange;
}
</style>
</head>

<body ng-app="firstApplication">
<div id="layoutContainer" ng-controller="layoutController as ctrl" layout="row"
style="height:100px;" ng-cloak layout-wrap>
<div flex="30" class="red box">
  [flex="30"]
</div>
<div flex="45" class="orange box">
  [flex="45"]
</div>
<div flex="25" class="red box">
  [flex="25"]
</div>
<div flex="33" class="red box">
  [flex="33"]
</div>
<div flex="66" class="orange box">
  [flex="66"]
</div>
<div flex="50" class="orange box">
  [flex="50"]
</div>
<div flex class="red box">
  [flex]
</div>
</div>
</body>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
angular.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
angular-animate.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
angular-aria.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
angular-messages.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angular_material/1.0.0/
angular-material.min.js"></script>
<script type="text/javascript">
angular
  .module('firstApplication', ['ngMaterial'])
  .controller('layoutController', layoutController);

  function layoutController ($scope) {
  }
</script>

</html>
```

- **md-search-text:** um objeto model do Angular para vincular ao texto de consulta da pesquisa, caso deseje usá-lo mais à frente dentro do código JavaScript do AngularJS;
- **md-selected-item:** um model do Angular para vincular ao item selecionado, caso deseje usá-lo mais à frente dentro do código JavaScript do AngularJS;
- **md-item-text:** uma expressão que irá converter o seu objeto em uma string única;
- **ng-disabled:** determina sim ou não para desativar o campo de entrada;
- **placeholder:** texto que será exibido no campo de input sob a forma de dica;
- **md-min-length:** especifica o comprimento mínimo do texto antes do preenchimento automático das sugestões.

O exemplo da **Listagem 8** (retirado da seção “Demos” da documentação oficial do Angular Material – vide seção **Links**) mostra como usar a diretiva *md-autocomplete*. Nele, veremos como criar um exemplo de caixa de pesquisa que buscará por um dos estados brasileiros (configurados em uma lista estática dentro do código JavaScript, criado mais adiante), com opções para habilitar um efeito de carregamento com uma barra de progresso, usando o recurso nativo de cache do próprio Angular, além habilitar ou desabilitar o campo de busca em si. Vejamos algumas de suas principais partes:

- As definições de imports, diretivas de aplicação e controller se mantêm aqui nessa listagem.
- Na linha 11 declaramos a diretiva `<md-autocomplete>` com vários parâmetros configurados para a mesma, a saber:
 - `ng-disabled`: configura o estado de campo habilitado ou não, tal valor será pego da checkbox mais abaixo;
 - `md-no-cache`: define se os dados do campo de busca serão salvos em cache no browser ou não;
 - `md-selected-item`: define qual o item que foi selecionado e, portanto, o *value* do campo em si;
 - `md-search-text-change`: define a função JavaScript a ser chamada quando o texto do campo for alterado (a mesma está definida na linha 29 da **Listagem 9** e apenas imprime em console o valor atual);
 - `md-search-text`: define o valor do texto no campo de busca;
 - `md-selected-item-change`: define a função JavaScript que será chamada quando um novo item for selecionado entre as demais opções na lista (cuja declaração foi feita na linha 32 da **Listagem 9** e também só imprime o valor no console);
 - `md-items`: define a iteração sobre os itens da lista que será retornada pela função `queryBusca()` a serem disponibilizados no mesmo campo;
 - `md-item-text`: define o texto de cada item a ser exibido na listagem de opções;
 - `md-min-length`: define o tamanho mínimo da lista de itens a ser exibida;
 - `placeholder`: a dica que aparecerá no campo antes que o usuário digite qualquer coisa.

- Na linha 22 definimos um elemento `<md-item-template>` que usaremos para dizer ao AngularJS qual a fonte de dados e como ele deve exibir cada elemento da listagem;
- Na linha 25 definimos via tag `<md-not-found>` o texto padrão a ser exibido quando nenhum item for encontrado para o texto informado no campo de busca;
- Nas linhas 31 a 34 definimos as opções dos checkboxes com suas respectivas ações para exibir ou não a barra de progresso que vimos, desabilitar o cache ou o respectivo campo de busca;

A segunda parte do código encontra-se na **Listagem 9**. O mesmo trecho deve ser inserido entre as tags `<script>` das linhas 44 e 46 da listagem anterior, pois trata-se do código que dinamizará a busca via Angular Material.

Na linha 4 temos a função que mapeia os principais fluxos de execução. Nas primeiras linhas declaramos as variáveis a serem usadas ao longo do código (note que a função `novoEstado` não tem implementação, é apenas uma referência para que possamos implementar essa adição dinâmica em um exemplo mais formal). A função `queryBusca` (linha 17) se encarrega de receber o objeto de query e fazer uma filtragem inicial para saber se já temos no nosso próprio cache (variável `self`). Veja que o código faz uso também de `promises` (linha 24) para lidar com o processamento assíncrono da listagem de itens. As `promises` disponibilizam meios de processamento assíncrono dentro do JavaScript que, por padrão, não suporta tal recurso. O Angular faz uso delas para tornar o componente de autocomplete assíncrono, uma vez que não é possível saber quanto tempo pode demorar uma requisição a uma lista de valores remota a ser exibida pelo mesmo. A função `loadEstados` (linha 36), por sua vez, se encarrega de criar uma string com todos os valores separados por vírgula para que possamos quebrá-la (`split`) e mapeá-la (`map`) com chaves-valores. A função `createFilterFor` (linha 46) mascara todos os valores para minúsculo para que possamos assim não ter diferenças entre o que está na lista de estados e o que está de fato na tela.

Veja na **Figura 7** como ficará nosso código final na tela.

O fato de o *md-autocomplete* exibir resultados em cache ao executar uma consulta permite que, após a primeira chamada, ele use os resultados em cache para eliminar solicitações ou lógicas de servidor desnecessárias, trazendo os resultados de imediato, como podemos ver na **Figura 8**.

Ele também pode ser desativado como mostra a **Figura 9**, onde o box fica bloqueado para uma nova digitação.

Botom Sheet

O botom sheet é um pequeno menu que desliza para cima a partir da parte inferior da tela, cobrindo o conteúdo e mantendo o foco. Originalmente destinado a ser utilizado exclusivamente para dispositivos móveis, o *botom sheet* vem ganhando popularidade em telas maiores. Para usá-lo, temos que criar um template com a diretiva *md-bottom-sheet*, que contém tanto uma classe *md-grid* como uma *md-list*, as quais são usadas para criar elementos de grade e lista no layout do Material, respectivamente.

Listagem 8. Implementado a diretiva autocomplete – Parte 1.

```
01 <html lang="pt">
02 <head>
03   <meta charset="utf-8"/>
04   <link rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/
    angular_material/1.0.0/angular-material.min.css">
05 </head>
06 <body ng-app="estadosApp" ng-cloak>
07   <div ng-controller="autoCompleteController as estApp"
      layout="column" ng-cloak>
08     <md-content class="md-padding">
09       <form ng-submit="$event.preventDefault()">
10
11       <md-autocomplete
12         ng-disabled="estApp.isDisabled"
13         md-no-cache="estApp.noCache"
14         md-selected-item="estApp.selectedItem"
15         md-search-text-change="estApp.searchTextChange
          (estApp.searchText)"
16         md-search-text="estApp.searchText"
17         md-selected-item-change="estApp.selectedItemChange(item)"
18         md-items="item in estApp.queryBusca(estApp.searchText)"
19         md-item-text="item.display"
20         md-min-length="0"
21         placeholder="Digite o nome do estado...">
22         <md-item-template>
23           <span md-highlight-text="estApp.searchText"
              md-highlight-flags="^i">{{item.display}}</span>
24         </md-item-template>
25         <md-not-found>
26           Nenhum resultado encontrado para "{{estApp.searchText}}".
27           <a ng-click="estApp.novoEstado(estApp.searchText)">
              Criar um novo!</a>
28         </md-not-found>
29       </md-autocomplete>
30       <br/>
31       <md-checkbox ng-model="estApp.similarQuery">
          Exibir barra de progresso?</md-checkbox>
32       <md-checkbox ng-model="estApp.noCache">Desabilitar cache?
            </md-checkbox>
33       <md-checkbox ng-model="estApp.isDisabled">
          Desabilitar campo de busca?</md-checkbox>
34
35     </form>
36   </md-content>
37 </div>
38 </body>
39 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular.min.js"></script>
40 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular-animate.min.js"></script>
41 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular-aria.min.js"></script>
42 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular-messages.min.js"></script>
43 <script src="http://ajax.googleapis.com/ajax/libs/angular_material/1.0.0/
  angular-material.min.js"></script>
44 <script type="text/javascript">
45   // Código do autocomplete
46 </script>
47 </html>
```

Listagem 9. Implementado a diretiva autocomplete – Parte 2.

```
01 angular.module('estadosApp', ['ngMaterial'])
02 .controller('autoCompleteController', autoCompleteController);
03
04 function autoCompleteController ($timeout, $q, $log) {
05   var self = this;
06   self.similarQuery = false;
07   self.isDisabled = false;
08   // lista de estados a serem exibidos
09   self.estados = loadEstados();
10   self.queryBusca = queryBusca;
11   self.selectedItemChange = selectedItemChange;
12   self.searchTextChange = searchTextChange;
13   self.novoEstado = novoEstado;
14   function novoEstado(state) {
15     alert("A implementar...");
```

```
16   }
17   function queryBusca (query) {
18     var results = query ? self.estados.filter( createFilterFor(query) )
        : self.estados, deferred;
19     if (self.similarQuery) {
20       deferred = $q.defer();
21       $timeout(function () {
22         deferred.resolve( results );
23       }, Math.random() * 1000, false);
24       return deferred.promise;
25     } else {
26       return results;
27     }
28   }
29   function searchTextChange(text) {
30     $log.info('Texto modificado para:' + text);
31   }
32   function selectedItemChange(item) {
33     $log.info('Item modificado para:' + JSON.stringify(item));
34   }
35   // constrói uma lista de estados como map de pares de chave-valor
36   function loadEstados() {
37     var allEstados = 'Acre (AC), Alagoas (AL), Amapá (AP), Amazonas (AM),
      Bahia (BA), Ceará (CE), Distrito Federal (DF), Espírito Santo (ES), Goiás (GO),
      Maranhão (MA), Mato Grosso (MT), Mato Grosso do Sul (MS),
      Minas Gerais (MG), Pará (PA), Paraíba (PB), Paraná (PR), Pernambuco (PE),
      Piauí (PI), Rio de Janeiro (RJ), Rio Grande do Norte (RN),
      Rio Grande do Sul (RS), Rondônia (RO), Roraima (RR), Santa Catarina (SC),
      São Paulo (SP), Sergipe (SE), Tocantins (TO)';
38     return allEstados.split(/,/).map(function (state) {
39       return {
40         value: state.toLowerCase(),
41         display: state
42       };
43     });
44   }
45   // filtra pela query de busca
46   function createFilterFor(query) {
47     var lowercaseQuery = angular.lowercase(query);
48     return function filterFn(state) {
49       return (state.value.indexOf(lowercaseQuery) === 0);
50     };
51   }
52 }
```

Em seguida, chamá-lo com o serviço do *bottom sheet*: o `$mdBottomSheet.show()`. A **Listagem 10** traz um exemplo prático de uso para essa diretiva. Veja que a nível de HTML nenhuma mudança significativa foi efetuada, apenas no JavaScript. A função `abrirBottomSheet()` se encarregou de receber o módulo do `$mdBottomSheet`

e chamar a sua função `show()`, passando o template com uma string em HTML a ser exibida (veja que é nela onde devem estar inseridas as tags `<md-bottom-sheet>`).

A **Figura 10** mostra como ficará o nosso botão criado através dessa diretiva, já na **Figura 11** podemos ver como ficará a tela após

o click no botão. Veja que a nova tela cobre o antigo conteúdo, mas ainda o exibe em plano de fundo.

Input

O *md-input-container* é um componente recipiente para conter qualquer elemento *<input>* ou *<textarea>* como um filho. O *md-input-container* também suporta a manipulação de erros usando o padrão do AngularJS, via diretivas *ng-messages*, e anima as mensagens usando os eventos de *ngEnter/ngLeave* ou os eventos de *ngShow/ngHide*.



Figura 7. Página com a diretiva autocomplete



Figura 8. Digitando um texto no box que contém a diretiva autocomplete e valor em cache



Figura 9. Desabilitando o box de busca



Figura 10. Exemplo do botão



Figura 11. Exibição da tela usando a diretiva bottom sheet

Listagem 10. Implementando a diretiva bottom sheet

```
01 <html lang="pt">
02 <head>
03   <meta charset="utf-8"/>
04   <link rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/
        angular_material/1.0.0/angular-material.min.css">
05 </head>
06 <body ng-app="bottomSheetApp">
07   <div ng-controller="bottomSheetController as ctrl" layout="column">
08     <md-content class="md-padding">
09       <form ng-submit="$event.preventDefault()">
10         <md-button class="md-raised md-primary" ng-click="abrirBottomSheet()">
11           Abrir Bottom Sheet!
12         </md-button>
13       </form>
14     </md-content>
15   </div>
16 </body>
17 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
        angular.min.js"></script>
18 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
        angular-animate.min.js"></script>
19 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
        angular-aria.min.js"></script>
20 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
        angular-messages.min.js"></script>
21 <script src="http://ajax.googleapis.com/ajax/libs/angular_material/1.0.0/
        angular-material.min.js"></script>
22 <script type="text/javascript">
23   angular
24     .module('bottomSheetApp', ['ngMaterial'])
25     .controller('bottomSheetController', bottomSheetController);
26
27   function bottomSheetController ($scope, $mdBottomSheet) {
28     $scope.abrirBottomSheet = function() {
29       $mdBottomSheet.show({
30         template:'<md-bottom-sheet>Aprender <b>Angular Material</b>!!
31           </md-bottom-sheet>'
32       });
33     }
34   </script>
35 </html>
```

Insira em alguns dos seus campos de texto a diretiva de *input*. Envolve sua tag de *input* com a diretiva *md-input-container* e você verá isso acontecer. A diretiva de *input* lida com um tipo de elemento no HTML que sempre foi tido como entediante, mas no fim proporciona uma agradável surpresa pela capacidade que o Angular Material fornece.

Como em várias diretivas apresentadas, o *input* também possui alguns atributos, a seguir iremos conhecer alguns e, logo mais na Listagem 11, iremos criar um formulário com essa diretiva.

- **md-maxlength:** limita o número máximo de caracteres permitidos na entrada. Se isso for especificado, um contador de caracteres será exibido na parte inferior da entrada. O objetivo do

`md-maxlength` é exatamente mostrar o contador do comprimento máximo do texto. Se você só precisa de uma validação simples, pode usar os atributos `ng-maxlength` ou `maxlength`:

- **aria-label:** é necessária quando nenhuma label está presente (uma mensagem de aviso será registrada no console quando isso ocorrer);
- **placeholder:** uma alternativa ao `aria-label` quando a label não está presente. O texto do `placeholder` é copiado para o atributo `aria-label`;
- **md-no-autogrow:** quando presente, áreas de texto não vão crescer automaticamente;

- **md-detect-hidden:** quando presente, as áreas de texto serão dimensionadas adequadamente quando forem reveladas após terem se escondido. Esse é desativado por padrão em vista de motivos de desempenho, pois garante um refluxo ao digerir cada ciclo.

Note que o próprio AngularJS já se encarrega de validar os atributos base da HTML5, como o `required` (para campo obrigatório) e o `maxlength`, por exemplo. Para cada campo devemos definir uma div com a diretiva `ng-messages`, que irá receber também um `role` (um papel específico dentro do Angular) referente ao tipo de mensagem que ali será exibida (o valor `alert` confere a cor amarela bem como configurações específicas do Angular Material). Na tag interna de diretiva `ng-message-exp` definimos quais validações deverão ser consideradas por padrão naquele campo em si e dentro

dela a mensagem que deverá ser exibida caso alguma retorne falso (linhas 21 a 24). Caso seja necessário exibir uma mensagem para cada regra de validação, podemos definir várias divs com uma `ng-message` para cada uma (linhas 31 e 32).

No JavaScript da linha 51, contudo, apenas definimos o texto que já deve vir preenchido no campo por padrão.

As **Figuras 12 a 14** mostram como ficará o formulário que acabamos de implementar.

Grid Lists

São uma alternativa às listas padrão. Uma *grid list* é melhor para a apresentação de imagens e é otimizada para uma melhor compreensão visual dos elementos. Ela funciona através da fixação de diferentes *tiles* (blocos de divs HTML) feitos sob medida em uma grid, dando uma sensação eclética à implementação. Vejamos na **Listagem 12** um exemplo simples fornecido pela documentação do Angular Material de tilesets (conjunto de blocos *tiles*) que geram vários quadros dinâmicos na página, com cores aleatórias. As propriedades:

- `md-cols*` definem valores para as larguras dos tiles;
- `md-row*` definem valores para as alturas dos tiles;
- `md-gutter*` definem valores para as margens entre cada um dos tiles;
- `md-colspan*` definem quantas colunas cada tile ocupará na página (semelhante à abordagem com tabelas).

Listagem 11. Implementação de formulário com a diretiva input

```
01 <html lang="pt">
02 <head>
03   <meta charset="utf-8" />
04   <link rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/
      angular_material/1.0.0/angular-material.min.css">
05 </head>
06 <body ng-app="inputApp">
07   <div id="inputContainer" ng-controller="inputController as ctrl" ng-cloak>
08     <md-content layout-padding>
09       <form name="formProjeto">
10         <md-input-container class="md-block">
11           <label>Usuário</label>
12           <input required name="usuario" ng-model="project.usuario">
13           <div ng-messages="formProjeto.usuario.$error">
14             <div ng-message="required">Obrigatório.</div>
15           </div>
16         </md-input-container>
17         <md-input-container class="md-block">
18           <label>Email</label>
19           <input required type="email" name="emailUsuario"
              ng-model="project.emailUsuario"
              minlength="10" maxlength="100" ng-pattern="/^.+@.+..+$/" />
20           <div ng-messages="formProjeto.emailUsuario.$error" role="alert">
21             <div ng-message-exp="['required', 'minlength', 'maxlength', 'pattern']">
22               Seu email deve conter entre 10 e 100 caracteres de tamanho e deve ser
23               um endereço válido.
24             </div>
25           </div>
26         </md-input-container>
27         <md-input-container class="md-block">
28           <label>Comentários</label>
29           <input md-maxlength="300" required name="comentarios"
              ng-model="project.comentarios">
```

```
30         <div ng-messages="formProjeto.comentarios.$error">
31           <div ng-message="required">Obrigatório.</div>
32           <div ng-message="md-maxlength">Os comentários devem ter menos
              de 300 caracteres.</div>
33         </div>
34       </md-input-container>
35     </form>
36   </md-content>
37 </div>
38 </body>
39 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular.min.js"></script>
40 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular-animate.min.js"></script>
41 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular-aria.min.js"></script>
42 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
  angular-messages.min.js"></script>
43 <script src="http://ajax.googleapis.com/ajax/libs/angular_material/1.0.0/
  angular-material.min.js"></script>
44 <script type="text/javascript">
45   angular
46     .module('inputApp', ['ngMaterial'])
47     .controller('inputController', inputController);
48
49   function inputController($scope) {
50     $scope.project = {
51       comentarios: 'Comentários',
52     };
53   }
54 </script>
55 </html>
```

Primeiros passos com o Angular Material

Listagem 12. Exemplo com tilesets randômicos no Angular Material.

```
01 <html lang="pt">
02   <head>
03     <meta charset="utf-8"/>
04     <link rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/
          angular_material/1.0.0/angular-material.min.css"/>
05   </head>
06   <body ng-app="gridApp">
07     <div ng-controller="GridListCtrl as appCtrl" ng-cloak="" 
          class="gridListDemoResponsiveUsage">
08       <md-content layout-padding="">
09         <md-grid-list md-cols-gt-md="12" md-cols-sm="3" md-cols-md="8">
10           <md-row-height-gt-md="1:1" md-row-height="4:3" md-gutter-gt-md="16px">
11             <md-grid-tile ng-repeat="tile in appCtrl.colorTiles" ng-style="{
12               'background': tile.color
13             }" md-colspan-gt-sm="{{tile.colspan}}"
14               md-rowspan-gt-sm="{{tile.rowspan}}">
15             </md-grid-tile>
16           </md-grid-list>
17         </md-content>
18       </div>
19     </body>
20     <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
          angular.min.js"></script>
21     <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
          angular-animate.min.js"></script>
22     <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
          angular-aria.min.js"></script>
23     <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/
          angular-messages.min.js"></script>
24     <script src="http://ajax.googleapis.com/ajax/libs/angular_material/1.0.0/
          angular-material.min.js"></script>
25     <script type="text/javascript">
26       angular
27         .module('gridApp', ['ngMaterial'])
28         .controller('GridListCtrl', function ($scope) {
29           var COLORS = ['#ffbeebe', '#ffcdd2', '#ef9a9a', '#e57373', '#ef5350', '#f44336',
30             '#e53935', '#d32f2f', '#c62828', '#b71c1c', '#ff8a80', '#ff5252', '#ff1744',
31             '#d50000', '#f8bbd0', '#f48fb1', '#f06292', '#ec407a', '#e91e63', '#d81b60',
32             '#c2185b', '#ad1457', '#880e4f', '#ff80ab', '#ff4081', '#f50057', '#c51162',
33             '#e1bee7', '#ce93d8', '#ba68c8', '#ab47bc', '#9c27b0', '#8e24aa', '#7b1fa2',
34             '#4a148c', '#ea80fc', '#e040fb', '#d500f9', '#aa00ff', '#ede7f6', '#d1c4e9',
35             '#b39ddb', '#9575cd', '#7e57c2', '#673ab7', '#5e35b1', '#4527a0', '#311b92',
36             '#b388ff', '#7c4dff', '#651fff', '#6200ea', '#c5cae9', '#9fa8da', '#7986cb',
37             '#5c6bc0', '#3f51b5', '#3949ab', '#3039f9', '#283593', '#1a237e', '#8c9eff',
38             '#536dfe', '#3d5afe', '#304ffe', '#e3f2fd', '#bbdefb', '#90caf9', '#64b5f6', '#42a5f5',
39             '#2196f3', '#1e88e5', '#1976d2', '#1565c0', '#0d47a1', '#82b1ff', '#448aff',
40             '#2979ff', '#2962ff', '#b3e5fc', '#81d4fa', '#4fc3f7', '#29b6f6', '#03a9f4', '#039be5',
41             '#0288d1', '#0277bd', '#01579b', '#80d8ff', '#40c4ff', '#00b0ff', '#0091ea,
42             '#e0f7fa', '#b2ebef', '#80deeaa', '#4dd0e1', '#26c6da', '#00bcd4', '#00acc1',
```

```
'#0097a7', '#00838f', '#006064', '#84ffff', '#18ffff', '#00e5ff', '#00b8d4', '#e0f2f1',
  '#b2dfdb', '#80cbc4', '#4db6ac', '#26a69a', '#009688', '#00897b', '#00796b',
  '#00695c', '#a7ffeb', '#64ffda', '#1de9b6', '#00bfa5', '#e8f5e9', '#c8e6c9',
  '#a5d6a7', '#81c784', '#66bb6a', '#4caf50', '#43a047', '#388e3c', '#2e7d32',
  '#1b5e20', '#b9f6ca', '#69f0ae', '#00e676', '#00c853', '#f1f8e9', '#dcedc8',
  '#c5e1a5', '#aed581', '#9ccc65', '#8bc34a', '#7cb342', '#689f38', '#558b2f',
  '#33691e', '#ccff90', '#b2ff59', '#76ff03', '#64dd17', '#9fbbe7', '#0f4f33', '#e6ee9c',
  '#dce775', '#d4e157', '#cddc39', '#c0ca33', '#afb42b', '#9e9d24', '#827717',
  '#f4ff81', '#e0ff41', '#c6ff00', '#aeee00', '#ffffde7', '#ffff9c4', '#ffff59d', '#ffff176',
  '#ffffe58', '#ffb3b', '#fdd835', '#fb02d', '#f9a825', '#f57f17', '#ffff8d', '#ffff00',
  '#ffe000', '#ffd600', '#fff8e1', '#ffecb3', '#ffe082', '#ffd54f', '#ffc28', '#ffc107',
  '#ffb300', '#ffa000', '#ff8f00', '#ff6f00', '#ffe57f', '#ffd740', '#ffc400', '#ffab00',
  '#ffff3e0', '#ffe0b2', '#ffcc80', '#ffb74d', '#ffa726', '#ff9800', '#fb8c00', '#f57c00',
  '#e6f6c00', '#e65100', '#ffd180', '#ffab40', '#ff9100', '#f6d00', '#fbe9e7', '#ffccbc',
  '#ffab91', '#ff8a65', '#ff7043', '#ff5722', '#f4511e', '#e64a19', '#d84315', '#bf360c',
  '#ffffe80', '#ff6e40', '#ff3d00', '#dd2c00', '#d7ccc8', '#bcaa4', '#795548',
  '#d7ccc8', '#bcaa4', '#8d6e63', '#eceff1', '#cf8dc', '#b0bec5', '#90a4ae',
  '#78909c', '#607d8b', '#546e7a', '#cf8dc', '#b0bec5', '#78909c];
```

28
29 this.colorTiles = (function () {
30 var tiles = [];
31 for (var i = 0; i < 46; i++) {
32 tiles.push({
33 color: randomColor(),
34 colspan: randomSpan(),
35 rowspan: randomSpan()
36 });
37 }
38 return tiles;
39 })();
40
41 function randomColor() {
42 return COLORS[Math.floor(Math.random() * COLORS.length)];
43 }
44
45 function randomSpan() {
46 var r = Math.random();
47 if (r < 0.8) {
48 return 1;
49 } else if (r < 0.9) {
50 return 2;
51 } else {
52 return 3;
53 }
54 }
55 }
56 });
57 </script>
58 </html>

Usuário

Email

Comentários

Comentários

11/300

Figura 12. Formulário com diretiva input

Usuário

Obrigatório.

Email

Comentários

Comentários

11/300

Figura 13. Formulário com mensagem de campo obrigatório exibida

Figura 14. Aqui podemos ver o funcionamento dos métodos `ng-message-exp` e `md-maxlength`

No JavaScript apenas criamos um vetor com várias cores em formato hexadecimal e as randomizamos via função `randomColor` (linha 42), da mesma forma que definimos `rowspan's` e `colspan's` distintos e aleatórios via função `randomSpan` (linha 46). Veja também que a função `colorTiles` da linha 29 faz uso dessas funções randômicas para gerar e atribuir as cores/dimensões referentes a cada `tile`. Se o leitor desejar fazer testes com mais tiles basta incrementar o valor que está sendo iterado no loop `for`.

O tamanho do `tile` e `layout`, então, corresponde ao tamanho da tela conforme ilustrado na **Figura 15**.

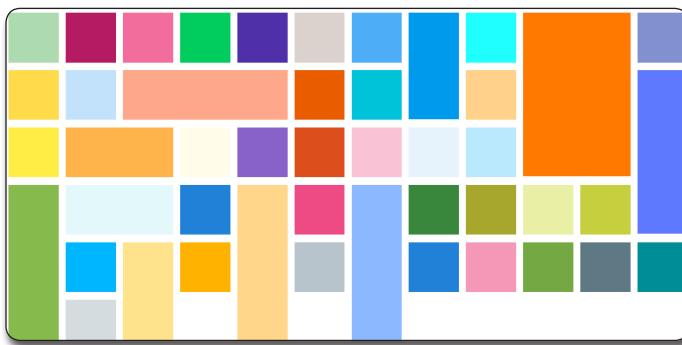


Figura 15. Exemplo final da grid list

O Google está convertendo suas aplicações mais populares para o Material Design. Agora eles estão trabalhando no desenvolvimento do Angular Material, uma implementação do Material Design escrito em AngularJS acrescentando sempre novos recursos e funcionalidades. Dessa forma você terá sempre em mãos o poder desses dois poderosos frameworks: um totalmente focado nos componentes/estilos da página e outro na dinamização desses de modo a não necessitarmos de todo um aparato de servidor como era feito antigamente.

Podemos notar nesse artigo que o Material Design e o Angular Material são uma fantástica maneira de aplicar as especificações de design para suas aplicações *single-page*. Se você quiser criar seu próprio aplicativo com o Angular Material, não perca tempo começando do zero. Em vez disso, comece com um aplicativo totalmente funcional conforme as demonstrações das diretivas configuradas. E claro, você pode aprender tudo sobre Angular Material visitando a documentação oficial na seção **Links**.

Autor



Fabrício Galdino

É um especialista em software e trabalhou com análise de TI e desenvolvimento de negócios por cinco anos. Tem experiência com testes e tecnologias relacionadas ao front-end, como SEO, Responsividade, HTML5 e CSS3.



Links:

Documentação Oficial do Angular Material

<https://material.angularjs.org/latest/>

Rails + Bootstrap: Integrando as tecnologias na criação de aplicações web

Aprenda a fazer uso do poder de ambos os frameworks para criar aplicações web ricas e flexíveis

ESTE ARTIGO É DO TIPO MENTORING

SAIBA MAIS: WWW.DEVMEDIA.COM.BR/MENTORING-SAIBAMAIIS

Bootstrap é um framework estruturado de forma a tornar mais fácil ao desenvolvedor criar um design agradável para sites ou aplicações web. Há classes CSS predefinidas para a criação de componentes comuns, tais como widgets, elementos de tipografia, listas, formulários e muitos outros. O framework também oferece funções e uma API JavaScript que torna mais fácil a criação de modelos, *popovers*, *scrollspy*, acordeões, entre inúmeros outros recursos comuns às aplicações web. A sua documentação é bem profunda, proporcionando exemplos de códigos para a maioria, se não para todos os componentes que fornece o Bootstrap.

O Bootstrap tem um grande guia de primeiros passos que o ajudará a integrá-lo em sua aplicação. E quando falamos em aplicações nos referimos exatamente a qualquer tipo de tecnologia *server side*: Java, Python ou Ruby, por exemplo.

O Rails, por sua vez, nasceu como uma extensão à linguagem de programação Ruby tendo como objetivo principal demandar todo o universo web da linguagem, já que a mesma não fornecia nenhum recurso nativo para

Cenário

Constantemente nos vemos forçados a lidar com novas tecnologias e, mais que isso, integrar as mesmas em um mesmo projeto usando recursos afins e determinando uma boa experiência final com o usuário. Um dos maiores desafios de efetuar esse tipo de procedimento são as incompatibilidades das tecnologias, bem como a falta de material oficial que auxilie nos passos de tal migração. Nesse artigo veremos como integrar as tecnologias do Rails com o famoso framework de componentes do Bootstrap através de uma aplicação web que abrace os principais recursos de ambos. Assim, você terá insumos suficientes para dar os primeiros passos e adaptar as suas aplicações para usar ambas as tecnologias.

tal finalidade. Trata-se especificamente de uma biblioteca que provê recursos para desenvolvimento usando o protocolo HTTP e o Ruby juntos. A riqueza do Rails é tanta que se expressa na grande quantidade de adeptos e na abrangência de suas implementações tanto em projetos e comunidades open source quanto nas grandes empresas que o usam por padrão para desenvolver suas soluções.

Uma das grandes dificuldades da maioria dos desenvolvedores que iniciam numa nova linguagem ou ambiente de programação consiste na integração de uma tecnologia à outra. Em vários pontos da web coletamos informações pertinentes sobre determinados frameworks (como a própria documentação oficial, por exemplo) que nos ajudam a entendê-lo praticamente por inteiro, porém, na

maioria das vezes, de forma individual, o que acaba não atendendo às necessidades de sistemas reais que os mesmos desenvolvedores lidarão no dia a dia.

O objetivo desse artigo é levá-lo através desse processo no que se refere à integração de ambos os frameworks em um projeto simples de CRUD de tarefas que nos ajudará a entender como as estruturas funcionam em conjunto. Assim, o leitor terá embasamentos suficientes para efetuar quaisquer outros procedimentos que envolvam o uso do Bootstrap e do Rails juntos em um ambiente web totalmente dinâmico.

Criando uma nova aplicação Rails

Vamos criar uma nova aplicação no Rails para que o leitor entenda todos os passos necessários para se atingir tal objetivo. Se você já possui um aplicativo que deseja integrar ao Bootstrap pode pular este passo.

No caso de estar usando o Windows, podemos proceder com a instalação usando o utilitário do *rubyinstaller* para Windows (na seção **Links** você encontra a URL de download do mesmo), mas também é possível configurar uma máquina virtual (VM) e, em seguida, seguir com as mesmas instruções. Portanto, baixe o instalador (verifique a versão correspondente ao seu Sistema Operacional – 32 ou 64 bits) e execute o mesmo seguindo os passos da instalação até o fim. Certifique-se de deixar marcada a opção “*Add Ruby executables to your PATH*” (tal como demonstrado na **Figura 1**) para garantir que os binários do Ruby sejam adicionados ao nosso caminho da variável de ambiente *Path* e, portanto, estejam disponíveis via linha de comando.

Para verificar se o Ruby foi realmente instalado, execute o seguinte comando no seu prompt cmd:

```
ruby -v
```

Em alguns ambientes, dependendo da configuração dos mesmos, pode se fazer necessária a instalação do *Development Kit* (disponível para download no mesmo link do RubyInstaller). Esse kit traz uma série de recursos e adaptações do Ruby para o ambiente do Windows e evita, assim, que alguns problemas de compatibilidade entre libs venham a atrapalhar o seu desenvolvimento. Portanto, para evitar isso efetue o download da versão zip mais recente e descompacte-a em um diretório de sua preferência. Depois, navegue via cmd até o referido diretório e execute os seguintes comandos:

```
ruby dk.rb init  
ruby dk.rb install
```

Isso será o suficiente para instalar o kit e associá-lo à sua variável de ambiente *Path*. Feito isso, vamos precisar nos certificar de instalar a *gem* (vide **BOX 1**) do *Rails*, executando o seguinte comando:

```
gem install rails
```

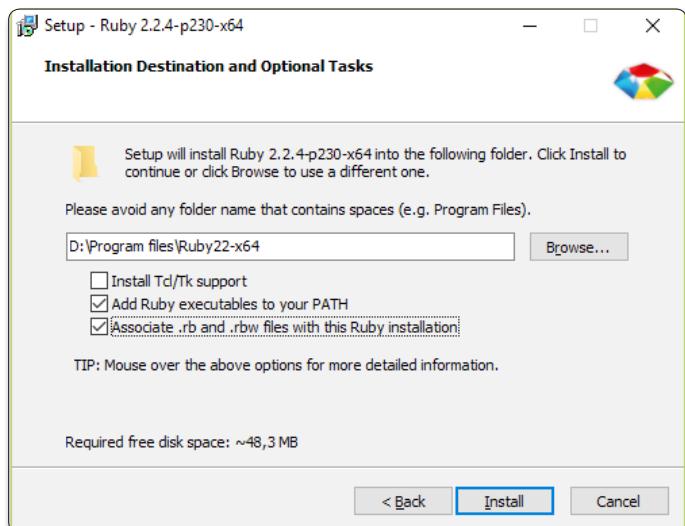


Figura 1. Marcando opções para adicionar Ruby ao Path

BOX 1. Gems

São espécies de pacotes disponibilizados pelo Ruby para trabalhar com diversas funcionalidades disponibilizadas para auxiliar no processo de desenvolvimento. O conceito é semelhante ao npm do Node.js.

Esse processo de download da gem do Rails pode demorar um pouco e não exibir nenhuma mensagem no cmd durante esse período, dando a impressão de que algo de errado ocorreu, mas não. Se quiser ter noção do que está acontecendo, execute o comando com a partícula *-V* no final, isso o forçará a ser mais verboso e imprimir mensagens mais claras do que está acontecendo. Agora vamos criar nossa aplicação do Rails via linha de comando, dando à mesma o nome de *tarefas_app*:

```
rails new tarefas_app
```

É importante estar dentro de um diretório que você vá usar de fato como workspace no cmd antes de executar o comando. Todo o processo de criação e gerenciamento dos arquivos do projeto pode ser feito pela interface de linha de comando, mas o leitor pode ficar à vontade para usar qualquer IDE de sua preferência. Aguarde até que o Rails efetue todo o processo de criação do projeto. Você verá, no final, uma estrutura de diretórios e arquivos que está explicada na **Tabela 1**.

Após isso, entre na pasta do projeto via comando *cd* para que possamos gerar um modelo no Rails chamado *Tarefa*, que representará uma tarefa a ser realizada:

```
rails generate scaffold Tarefa titulo:string descricao:text
```

Se por acaso alguma exceção relacionada a libs não disponíveis aparecer, siga os passos sugeridos pelo próprio Ruby para ajustar. Veja que o comando já define os atributos que o objeto *Tarefa* terá por padrão: um campo *titulo* cujo tipo será *string* e um campo

descrição de tipo text. Em seguida, execute o seguinte comando que efetua a migração das alterações feitas nos arquivos Ruby para o nosso banco de dados:

```
rake db:migrate
```

| Arquivo/Diretório | Descrição |
|-------------------------|---|
| app/ | Contém os controllers, models, views, helpers, mailers e assets para a aplicação. |
| bin/ | Contém os scripts rails que iniciam a aplicação e pode conter outros scripts que você use para setup, deploy ou execução da sua aplicação. |
| config/ | Configura as rotas da sua aplicação, banco de dados e mais. |
| db/ | Contém o esquema atual do seu banco de dados, assim como as migrações de banco efetuados. |
| Gemfile Gemfile.lock | Esses arquivos nos permitem especificar quais gems dependentes são necessárias à nossa aplicação Rails e quais estão bloqueadas, respectivamente. |
| lib/ | Módulos estendidos para a aplicação. |
| public/ | O único diretório aberto para a web, contém arquivos estáticos e assets compilados. |
| test/ | Testes unitários, mocks e outros aparatos de testes. |
| tmp/ | Arquivos temporários: cache, pid, arquivos de sessão, etc. |
| vendor/ | Lugar para todas as dependências de terceiros. Geralmente incluímos apenas gems de outros. |

Tabela 1. Lista de arquivos/diretórios criados na aplicação rails

O comando faz uso do *Rake*, um utilitário do Ruby usado de forma similar ao Ant para executar tarefas customizadas dentro do seu projeto Ruby. O comando, especificamente, reflete todas as nossas alterações nos objetos e código do projeto para o nosso banco de dados. Ao final de sua execução, você verá uma mensagem no cmd exibindo os objetos da base gerados/alterados. Nós precisamos de alguns dados inseridos nessa base para mostrar na nossa aplicação de tarefas, para fazer isso vamos “semear” (*seed*) o banco de dados com alguns dados de amostra. Coloque as seguintes linhas em seu arquivo *db/seeds.rb* conforme a **Listagem 1**. Esse arquivo basicamente se encarrega de alimentar a nossa base com dados de teste a serem usados pela aplicação.

Em seguida, para que os dados sejam adicionados de fato às tabelas execute em sua máquina o seguinte comando no prompt:

```
rake db:seed
```

Uma vez com os dados ok, vamos configurar uma rota simples no arquivo *config/routes.rb*, adicionando o valor *root 'tarefas#index'*, conforme mostra a **Listagem 2**. Trata-se da forma padrão como o Rails lida com o direcionamento de páginas e métodos dos controles posteriormente publicados em um servidor.

Agora, para testar tudo que fizemos até então, basta executar o servidor que o Rails disponibiliza internamente por padrão via comando:

```
rails s
```

Listagem 1. Inserindo dados de amostra na base.

```
# This file should contain all the record creation needed to seed the database with its default values.  
# The data can then be loaded with the rake db:seed (or created alongside the db with db:setup).  
  
#  
Tarefa.create!(titulo: 'fazer compras', descricao:'ovos, macarrão, açucar')  
Tarefa.create!(titulo: 'lavar o carro')  
Tarefa.create!(titulo: 'passar com o cachorro', descricao: 'No mínimo 30 minutos')  
Tarefa.create!(titulo: 'fazer revisão do carro', descricao: 'Na oficina do Seu João')  
Tarefa.create!(titulo: 'fazer o almoço', descricao: 'Macarronada com fritas')
```

Listagem 2. Configurando rota para acessarmos a funcionalidade de tarefas na web.

```
Rails.application.routes.draw do  
  resources :tarefas  
  root 'tarefas#index'  
end
```

Você verá uma mensagem do Rails inicializando o mesmo na porta 3000 em ambiente de localhost. Isso será mais que o suficiente para mapearmos o funcionamento da aplicação, além de nos mostrar as mensagens de exceção caso algo de errado aconteça no processo. Veja que ao iniciar o servidor o seu cmd estará a partir de agora ocupado servindo as aplicações, logo a sua janela do prompt não poderá ser usada para outras finalidades. Você pode abrir uma nova janela ou parar o servidor na atual (via Ctrl + C) para liberar o cmd. Quando você visitar o app em seu navegador, no endereço <http://localhost:3000>, deverá visualizar o que mostra a **Figura 2**.

Listing Tarefas

| Titulo | Descrição | |
|------------------------|------------------------|--|
| fazer compras | ovos, macarrão, açucar | <a>Show <a>Edit <a>Destroy |
| lavar o carro | | <a>Show <a>Edit <a>Destroy |
| passar com o cachorro | No mínimo 30 minutos | <a>Show <a>Edit <a>Destroy |
| fazer revisão do carro | Na oficina do Seu João | <a>Show <a>Edit <a>Destroy |
| fazer o almoço | Macarronada com fritas | <a>Show <a>Edit <a>Destroy |

New Tarefa

Figura 2. Tela com listagem de tarefas e modelo padrão do Rails

Escolhendo um pré-processador de CSS

Quando você estiver usando uma estrutura como o Bootstrap, um pré-processador de CSS deve ser envolvido como forma de flexibilizar o formato do desenvolvimento, estilo e estruturação das páginas. Eles melhoram o CSS e adicionam funcionalidades, tais como o uso de variáveis, mixins e de hierarquia de regras. Vejamos três ótimas opções de pré-processadores CSS populares que podemos usar no nosso projeto.

LESS

O Bootstrap está escrito originalmente em LESS (você pode acessar a página oficial na seção [Links](#)).

Vejamos na **Listagem 3** um exemplo de LESS, extraído da página oficial, que quando compilado gerará um código conforme a **Listagem 4**. Esse tipo de abordagem é muito importante quando lidamos com o Bootstrap e é necessário conhecer a forma como o framework aborda a geração e distribuição dos arquivos.

Listagem 3. Exemplo oficial de uso do LESS.

```
@base: #f938ab;

.box-shadow(@style, @c) when (iscolor(@c)) {
  -webkit-box-shadow: @style @c;
  box-shadow: @style @c;
}
.box-shadow(@style, @alpha) when (isnumber(@alpha)) {
  .box-shadow(@style, rgba(0, 0, 0, @alpha));
}
.box {
  color: saturate(@base, 5%);
  border-color: lighten(@base, 30%);
  div { .box-shadow(0 0 5px, 30%) }
}
```

Listagem 4. Resultado da execução do LESS em questão.

```
.box {
  color: #fe33ac;
  border-color: #fdcdea;
}
.box div {
  -webkit-box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
  box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
}
```

SASS

Outro pré-processador utilizado é o SASS, um pré-processador de código CSS semelhante ao LESS, porém baseado em regras aninhadas para incutir os estilos das páginas.

O SASS usa espaços em branco para aninhar as regras do CSS. Além disso, não há lugar para o ponto e vírgula no framework, já que ele trabalha com o conceito de endentação do código para identificar cada parte no todo, conforme exemplo da **Listagem 5**, que compila conforme a **Listagem 6**.

SCSS (Sassy CSS)

O SCSS se assemelha muito ao código CSS padrão. Na verdade, você realmente pode mudar a extensão de um arquivo CSS para SCSS, executá-lo através de um pré-processador e ele irá compilar normalmente. Uma grande diferença é que, embora você possa escrever o mesmo código CSS que já esteja acostumado a fazer, você também pode aninhar as suas regras usando o modelo original do SASS. O mesmo exemplo pode ser escrito em SCSS tal como vemos na **Listagem 7**, e ainda ser compilado conforme a **Listagem 8**.

Nesse artigo vamos usar o SASS por padrão para construir o estilo das nossas páginas. Embora o Bootstrap seja escrito oficialmente usando o LESS, ele foi portado para o SASS e a versão SASS é mantida oficialmente pela equipe do framework.

Listagem 5. Exemplo de código simples no SASS.

```
.header
  display: block
  background-color: red

.image
  float: left
  max-width: 80px
```

Listagem 6. CSS gerado como resultado da execução do código anterior.

```
.header {
  display: block;
  background-color: red;
}

.header .image {
  float: left;
  max-width: 80px;
```

Listagem 7. Exemplo de mesmo código no SCSS.

```
.header {
  display: block;
  background-color: red;

  .image {
    float: left;
    max-width: 80px;
  }
}
```

Listagem 8. Resultado CSS do código em SCSS.

```
.header {
  display: block;
  background-color: red;

  .image {
    float: left;
    max-width: 80px;
  }
}
```

Se você estiver interessado em usar uma versão LESS, poderá seguir os detalhes disponibilizados pela documentação oficial para resolver conflitos ou sanar quaisquer dúvidas sobre o mesmo.

Integrando o Bootstrap com Rails

Agora que temos uma aplicação Rails com alguns dados, podemos começar a trabalhar na integração do Bootstrap junto à referida aplicação.

Para isso, precisamos instalar as *gems* do Ruby para o Bootstrap, tal como demonstrado na **Listagem 9**. Para tanto, dentro do diretório raiz do projeto, identifique o arquivo *Gemfile* e o abra com algum editor de texto qualquer. Sua estrutura interna vem repleta de exemplos de como inserir recursos e de como habilitar ou desabilitar diversas *gems* para seus projetos. Em nosso caso, inclua o código tal como instruído pela listagem, observando a observação posta logo após: por padrão, as novas versões do Rails já trazem a gem do sass-rails inclusa no momento da criação de um novo projeto, logo não se faz necessário descomentar a linha

de uso da mesma. Caso contrário, se você estiver usando uma versão mais antiga do Rails e a referida gem não existir no arquivo, então descomente a linha em questão.

Observe que também podemos referenciar os números das versões que queremos das *gems*. Isso ajuda na hora de gerenciar nossas dependências, e o processo é muito semelhante ao que temos nas outras linguagens de servidor. Se, no momento em que estiver lendo esse artigo, a versão do *bootstrap-sass* for superior à que estamos aqui configurando, fique à vontade para atualizá-la de igual modo.

Listagem 9. Configurando gems do Bootstrap no projeto.

```
source 'https://rubygems.org'  
...  
  
gem 'bootstrap-sass', '~> 3.3.6'  
gem 'autoprefixer-rails'  
  
# NOTE: The sass-rails gem is included with new Rails applications by default.  
# Please make sure that it is not already in your Gemfile before uncommenting it.  
# gem 'sass-rails'
```

O *Autoprefixer (autoprefixer-rails)* é opcional, mas recomendado. Ele adiciona automaticamente os prefixos de fornecedores distintos ao seu código CSS quando ele é compilado para evitar problemas de compatibilidade entre diferentes versões de browsers. Agora execute o comando *bundle install* para instalar as *gems*.

Por fim, vamos renomear o arquivo autogerado do *app/assets/stylesheets/application.css* para *app/assets/stylesheets/application.css.sass* para que o mesmo reconheça os comandos do SASS que iremos executar. Quando fizermos isso, poderemos então importar as demais dependências do Bootstrap no recém-renomeado arquivo:

```
@import "bootstrap-sprockets"  
@import "bootstrap"
```

Quando você compilar esse código, os *assets* importados renderizarão vários arquivos também compilados com seus conteúdos traduzidos para CSS e JavaScript nativos, mantendo também a declaração dos *@import* traduzidos para código inteligível pelos browsers.

Em seguida, para lidar com os *assets* de importação do código JavaScript do Bootstrap, precisamos adicionar o trecho `//= require bootstrap-sprockets` ao seu arquivo *app/assets/javascripts/application.js*, isso porque é o JavaScript quem lidera a importação e referência de arquivos dentro da aplicação via Bootstrap. É importante que essa declaração seja feita logo após a do `//= require jquery`, uma vez que o JavaScript do Bootstrap tem dependência direta dos arquivos do jQuery. O seu arquivo deve ficar bem semelhante ao da **Listagem 10**.

De igual modo, é também importante que a sentença `//= require_tree` seja a última coisa a ser importada no arquivo, pois é ela que carregará a árvore de dependências e precisará ter todas as outras carregadas antes dela para funcionar. A razão é que,

`//= require_tree` compila cada um dos outros arquivos JavaScript no diretório javascript e quaisquer subdiretórios. Se você importar o *bootstraps-sprockets* no final do código, por exemplo, seus outros scripts podem não ter acesso às funções de inicialização da árvore de herança do Rails.

Lembre-se também de reiniciar sempre o servidor após quaisquer alterações nos arquivos do *Gemfile*.

Listagem 10. Importação dos arquivos de scripts no projeto.

```
//= require jquery  
//= require jquery_ujs  
//= require turbolinks  
//= require bootstrap-sprockets  
//= require_tree .
```

A Grid

A fim de fazer uso da grid do Bootstrap, você precisa de três componentes em uma ordem muito específica. Em primeiro lugar, você deve ter um elemento (geralmente uma *div*), o qual deve ter a classe CSS *container*. Dentro dele, você precisa de um outro elemento de classe CSS *row*. Os únicos elementos que podem estar diretamente dentro dos *row* são os elementos *column*.

Na **Listagem 11** temos um exemplo HTML de um layout de grid básico em ação. Veja que o mesmo traz o uso das classes divisórias do Bootstrap, com suas conotações de tamanho de tela e divisão em colunas num total de doze. Não entraremos em maiores detalhes sobre as especificidades do Bootstrap para não perdermos o foco do artigo.

Listagem 11. Exemplo HTML básico usando a grid.

```
<html>  
  <body>  
    <div id="main-container" class="container">  
      <div class="row">  
        <div id="sidebar" class="col-xs-3">  
          <!-- sidebar content -->  
        </div>  
        <div id="content" class="col-xs-9">  
          <!-- main content -->  
        </div>  
      </div>  
    </div>  
  </body>  
</html>
```

Grid com layouts fluídos

Se você prefere que o seu projeto preencha toda a largura do navegador, o Bootstrap te fornece uma ótima opção para isso. Trata-se dos contêineres fluidos, onde você simplesmente muda o seu elemento exterior de classe CSS *container-classeCss* para passar a ter uma classe CSS *container-fluid* no lugar, e esse elemento irá fluir ao longo de todo o caminho até a largura total do navegador. Na **Listagem 12** vemos o mesmo exemplo da listagem anterior, mas com uma disposição bem mais fluída.

Inserindo grids na aplicação

Podemos melhorar o nosso aplicativo através da implementação do sistema de grids. Vamos criar o nosso container *div*, uma linha, e duas colunas diretamente no nosso arquivo autogerado pelo Rails *app/views/layouts/application.html.erb* (vide **Listagem 13**).

Listagem 12. Mesmo exemplo seguindo o design fluido.

```
<html>
<body>
<div id="main-container" class="container-fluid">
<div class="row">
<div id="sidebar" class="col-xs-3">
<!-- sidebar content -->
</div>
<div id="main-content" class="col-xs-9">
<!-- main content -->
</div>
</div>
</div>
</body>
</html>
```

Listagem 13. Conteúdo da nossa tela da aplicação.

```
<!DOCTYPE html>
<html>
<head>
<head>
<title>TarefasApp</title>
<%= stylesheet_link_tag 'application', media:'all', 'data-turbolinks-track' =>
  true %>
<%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
<%= csrf_meta_tags %>
</head>
</head>
<body>

<div id="main-container" class="container">
<div class="row">
<div class="col-xs-3">
<h3>Aplicativo de Tarefas</h3>
</div>
<div class="col-xs-9">
<%= yield %>
</div>
</div>
</div>

</body>
</html>
```

Observe que nós adicionamos uma coluna na barra lateral (*sidebar*) que tem três colunas de largura. Agora se você visitar a sua aplicação no seu navegador verá o conteúdo deslocado para a direita, abrindo espaço para a barra lateral à esquerda, que no momento só consta do título que demos a ela. Não esqueça de reiniciar o servidor no console antes de efetuar esse teste. Faça um teste rápido também: abra o código fonte da página no próprio navegador e verifique que o Rails agora importou todos os arquivos, tanto CSS quanto JavaScript, do Bootstrap na mesma.

A propósito, você pode aninhar suas linhas dentro de outras colunas, desde que cada linha não tenha mais que 12 colunas

especificadas diretamente dentro dela. Por exemplo, na **Listagem 14**, seria perfeitamente válido referenciar tal divisão interna de colunas e linhas, uma vez que o somatório total, no fim, não passará de 12.

Se você tivesse esse mesmo layout construído com algumas bordas coloridas aplicadas às suas *div's*, a **Figura 3** então representaria sua visualização final, com cada um dos itens internos aos outros ainda obedecendo à regra dos 12.

Listagem 14. Exemplo de layout dividido em um total de 12.

```
<div class="container">
<div class="row">
<div class="col-xs-3"></div>
<div class="col-xs-9">
<div class="row">
<div class="col-xs-6"></div>
<div class="row">
<div class="col-xs-6"></div>
<div class="col-xs-6"></div>
</div>
</div>
<div class="col-xs-6"></div>
<div class="row">
<div class="col-xs-6"></div>
<div class="col-xs-6"></div>
</div>
</div>
</div>
<div class="col-xs-6"></div>
<div class="row">
<div class="col-xs-6"></div>
<div class="col-xs-6"></div>
</div>
</div>
</div>
</div>
```

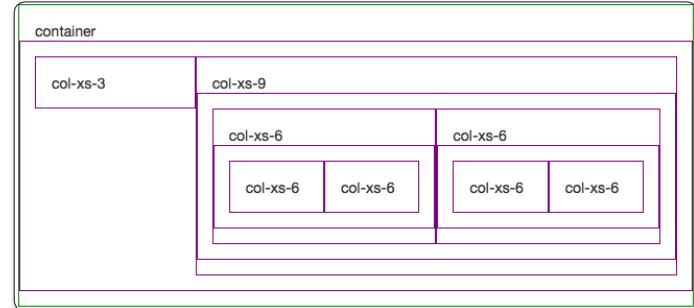


Figura 3. Exemplo de grids divididas com base na regra dos 12 do Bootstrap

Tabelas

As tabelas são um grande exemplo de como é fácil estilizar seus componentes usando classes CSS do Bootstrap. Tudo que você precisa fazer é adicionar a classe *table* para o elemento de tabela e atualizar a página: *<table class="table"> </table>*.

Há mais cores disponíveis para as tabelas, as quais são utilizadas anexando uma das seguintes classes:

- *table-striped*;
- *table-bordered*;
- *table-hover*;
- *table-condensed*;
- *table-responsive*.

Você também pode aplicar um estilo específico às linhas usando as classes de acompanhamento (aplicada ao elemento *tr*):

- *active*;
- *success*;
- *info*;
- *warning*;
- *danger*.

Tome um minuto para entender melhor cada uma dessas classes e se familiarizar com elas.

Botões

Criar um botão a partir de um link de texto é tão fácil como aplicar a classe *btn* para ele. Se você tem um botão e quiser criar um link de texto a partir dele, simplesmente adicione a classe *btn-link*.

As seguintes classes podem ser usadas em conjunto com a classe *btn* para criar o botão que você está procurando:

- *btn-default*;
- *btn-primary*;
- *btn-success*;
- *btn-info*;
- *btn-warning*;
- *btn-danger*;
- *btn-link*.

Você também pode alterar o tamanho do seu botão adicionando uma destas classes:

- *btn-lg* (*Large – Grande*);
- *btn-sm* (*Small – Pequeno*);
- *btn-xs* – (*X-Small – Extra pequeno*).

Você pode notar que o tamanho médio é deixado de fora, porque o meio é o tamanho padrão do botão, logo simplesmente não inclua qualquer uma dessas classes se desejar usar o tamanho padrão.

Para entender melhor como essa estrutura funciona, vamos criar um botão de sucesso, muito comum em aplicações desse tipo. Em seu arquivo *app/views/todos/index.html.erb*, adicione as classes *btn* e *btn-success* ao link “*New Tarefa*” (certifique-se de traduzir aos poucos os textos autogerados pelo Rails):

```
<%= link_to 'Nova Tarefa', new_tarefa_path, class:'btn btn-success' %>
```

Injetando estilo do Bootstrap

Para deixar o nosso código mais organizado, vamos efetuar a padronização da tela como um todo usando os estilos do Bootstrap. Mas antes, certifique-se de abrir a mesma numa aba do browser para que possamos comparar a transformação do nosso layout com o uso do Bootstrap. Na **Listagem 15**, temos o código da página *index.html.erb* como um todo atualizado, portanto modifique o seu conteúdo tal qual.

Note que os recursos do Rails já vêm preenchidos na tela, o que vamos fazer é essencialmente atuar sobre as classes CSS

do Bootstrap na página. Na linha 5 temos a inclusão das classes de tabela que vimos há pouco, em hierarquia: é necessário no mínimo importar a classe *table* para algum estilo e organização serem associados à mesma, os demais servem para incutir outros efeitos, como borda, efeito zebra para cada linha e o layout responsivo que se contrai e expande de acordo com o dispositivo onde a mesma for carregada. A estrutura da tabela, contudo, se mantém a mesma.

Listagem 15. Código final da página index da aplicação de tarefas.

```
01 <p id="notice"><%= notice %></p>
02
03 <h1>Lista de Tarefas</h1>
04
05 <table class="table table-bordered table-striped table-responsive">
06   <thead>
07     <tr>
08       <th>Título</th>
09       <th>Descrição</th>
10       <th colspan="3">Ações</th>
11     </tr>
12   </thead>
13
14   <tbody>
15     <% @tarefas.each do |tarefa| %>
16       <tr>
17         <td><%= tarefa.titulo %></td>
18         <td><%= tarefa.descricao %></td>
19         <td><%= link_to 'Exibir', tarefa, class:'btn btn-primary btn-xs' %></td>
20         <td><%= link_to 'Editar', edit_tarefa_path(tarefa), class:'btn btn-primary btn-xs' %></td>
21         <td><%= link_to 'Remover', tarefa, class:'btn btn-primary btn-xs', method: :delete, data: { confirm:'Tem certeza que deseja remover essa tarefa?' } %></td>
22       </tr>
23     <% end %>
24   </tbody>
25 </table>
26
27 <br>
28
29 <%= link_to 'Nova Tarefa', new_tarefa_path, class:'btn btn-success' %>
```

No loop de impressão dos resultados de cada tarefa com seus respectivos botões (linhas 15 a 23) temos os três links (agora traduzidos) de visualização, edição e remoção de cada item que agora contam com um novo atributo *class* declarado em suas inicializações. No fim, na linha 29, temos o botão que falamos há pouco de adição de novas tarefas.

O resultado da nossa tela pode ser visualizado na **Figura 4**.

Glyphicon

É incrível o quanto um ícone simples pode melhorar a aparência de um link ou botão, bem como outros componentes da sua interface gráfica. Você pode adicionar ícones para sua aplicação de maneira fácil, usando a biblioteca do *Glyphicon* (vide documentação do Bootstrap na seção **Links** para link oficial), uma biblioteca padrão de ícones que o Bootstrap disponibiliza gratuitamente em seu núcleo e que já vem autoinicializada por padrão junto ao Rails.



Figura 4. Tela com novos efeitos de estilo do Bootstrap

Vamos adicionar um ícone + ao botão “Nova Tarefa” e criar um novo botão de Cancelar ao lado dele para analisar como a biblioteca de ícones funciona. Além disso, modificaremos o conteúdo interno de cada um dos links de ações da tabela para ícones relacionados, de acordo com o código demonstrado na **Listagem 16**. Veja que cada um dos componentes de *link* do Rails se transforma agora em uma tag de início e fim bem definidos, com corpo interno. Os ícones devem ser referenciados via classe CSS dentro de elementos *<i>* da HTML. Também devemos remover os textos de cada um deles, exceto pelos de ação do fim da página, que devem especificar o que fazem quando clicados. Neste caso, especificamente, o texto migra para dentro do componente link após o ícone (ou depois, dependendo de como desejar exibi-lo).

Veja na **Figura 5** o resultado dessa tela após as alterações.



Figura 5. Tela com ícones nos botões da página

Formulários

Há uma tonelada de classes auxiliares para a inclusão de formulários em nossas páginas HTML. Veremos alguns exemplos básicos mais adiante.

A estrutura básica é muito simples. Cada par de label e controle dentro do elemento de formulários (*form*) são aninhados dentro de uma *div* com a classe *form-group*. Os controles são então atribuídos à classe *form-control*. Vejamos um exemplo bem simples dessa implementação na **Listagem 17**.

Podemos visualizar na **Figura 6** o resultado da implementação desse formulário.

Se você quiser mudar a sua forma e colocar o formulário em apenas uma linha, basta adicionar a classe *form-inline* ao elemento do seu formulário. Veja o resultado na **Figura 7**.

Listagem 16. Código da página index com ícones Glyphicons.

```

<p id="notice"><%= notice %></p>

<h1>Lista de Tarefas</h1>

<table class="table table-bordered table-striped table-responsive">
<thead>
<tr>
<th>Título</th>
<th>Descrição</th>
<th colspan="3">Ações</th>
</tr>
</thead>

<tbody>
<% @tarefas.each do |tarefa| %>
<tr>
<td><%= tarefa.titulo %></td>
<td><%= tarefa.descricao %></td>
<td align="center">
<%= link_to tarefa, class: 'btn btn-primary btn-xs' do %>
<i class="glyphicon glyphicon-eye-open"></i>
<% end %>
</td>
<td align="center">
<%= link_to edit_tarefa_path(tarefa), class: 'btn btn-primary btn-xs' do %>
<i class="glyphicon glyphicon-edit"></i>
<% end %>
</td>
<td align="center">
<%= link_to tarefa, class: 'btn btn-primary btn-xs', method: :delete,
data: { confirm: 'Tem certeza que deseja remover essa tarefa?' } do %>
<i class="glyphicon glyphicon-trash"></i>
<% end %>
</td>
</tr>
<% end %>
</tbody>
</table>

<br>

<%= link_to new_tarefa_path, class: 'btn btn-success' do %>
<i class="glyphicon glyphicon-plus"></i> Nova Tarefa
<% end %>

<%= link_to tarefas_path, class: 'btn btn-warning' do %>
<i class="glyphicon glyphicon-remove"></i> Cancelar
<% end %>

```

Listagem 17. Exemplo de formulário simples no Bootstrap.

```

<form role="form">
<div class="form-group">
<label for="usuario">Usuário</label>
<input type="text" id="usuario" name="usuario" class="form-control"/>
</div>
<div class="form-group">
<label for="senha">Senha</label>
<input type="password" id="senha" name="senha" class="form-control"/>
</div>
<input type="submit" value="Login"/>
<input type="button" value="Cancelar"/>
</form>

```

Rails + Bootstrap: Integrando as tecnologias na criação de aplicações web

```
<form role="form" class="form-inline">  
</form>
```

Outro estilo muito popular é a forma horizontal, onde a label fica à esquerda do controle, porém alinhada à direita. A fim de alcançar isso, precisamos usar o conhecimento sobre *grid's* que aprendemos anteriormente.

A label deve ser atribuída a um número de colunas predeterminado, e também deve ser configurada com a classe *control-label*:

```
<label for="username" class="col-sm-2 control-label">Usuário</label>
```

A seguir temos o código que aninha o controle dentro de uma *div* que especifica quantas colunas a parte direita do formulário deve tomar:

```
<div class="col-sm-10">  
<input type="text" id="username" name="username" class="form-control"/>  
</div>
```



Figura 6. Resultado do formulário simples



Figura 7. Mesmo formulário em uma só linha

Seus botões devem ser colocados dentro de uma *div* que especifica quantas colunas a mesma deve tomar. De um modo geral, o mesmo número deve ser igual ao dos inputs. Essa *div* deve, então, ser envolvida por outra *div*, de classe *form-group*. A única coisa diferente sobre seus botões é que eles não possuem uma label externa. Isso significa que você precisa usar uma classe de deslocamento para mover os botões de acordo com os controles. A **Listagem 18** mostra esse form atualizado e a **Figura 8** mostra como o mesmo será exibido.

A maioria dos tipos comuns de inputs são suportados, incluindo todos os tipos da HTML5. Caixas de seleção, botões de rádio, controles de texto estático e *selects* também são suportados de igual modo. A caixa de seleção “Lembrar de mim” (**Figura 9**) pode ser adicionada, incutindo o código da **Listagem 19** ao nosso código, acima dos botões.

Os botões de rádio podem ser adicionados através da *div radio*, vejamos um exemplo na **Listagem 20** (resultado na **Figura 10**).

Caixas de marcação e botões de rádio podem ser desativados normalmente via atributo HTML *disabled* diretamente no input, bem como via atribuição do container (que tem uma classe de checkbox ou rádio) da classe *disabled*.

Listagem 18. Formulário com classes de elementos agrupadas.

```
<form role="form" class="form-horizontal">  
<div class="form-group">  
<label for="username" class="col-sm-2 control-label">Username</label>  
<div class="col-sm-10">  
<input type="text" id="username" name="username" class="form-control"/>  
</div>  
</div>  
<div class="form-group">  
<label for="password" class="col-sm-2 control-label">Password</label>  
<div class="col-sm-10">  
<input type="password" id="password" name="password" class="form-control"/>  
</div>  
</div>  
  
<div class="form-group">  
<div class="col-sm-offset-2 col-sm-10">  
<input type="submit" value="Login" />  
<input type="button" value="Cancel" />  
</div>  
</div>  
</form>
```

Listagem 19. Checkbox para lembrar o usuário na aplicação.

```
<div class="form-group">  
<div class="col-sm-offset-2 col-sm-10">  
<div class="checkbox">  
<label>  
<input type="checkbox" value="1">  
Lembrar de mim  
</label>  
</div>  
</div>  
</div>
```

Listagem 20. Criando tela simples com botões de input checkbox.

```
<div class="form-group">  
<div class="col-sm-offset-2 col-sm-2">  
<div class="radio">  
<label>  
<input type="radio" name="radioLogin" id="JaExiste" value="usuario_existente" checked>  
Usuário já existe  
</label>  
</div>  
<div class="radio">  
<label>  
<input type="radio" name="radioLogin" id="novoUsuario" value="novo_usuario">  
Novo usuário  
</label>  
</div>  
</div>  
</div>
```

Em relação aos selects (combobox), basta atribuir a classe *form-control* ao elemento *<select>* e você automaticamente vai notar a aplicação do estilão padrão do Bootstrap.

Há muitas funcionalidades relativas a formulários incluídas no Bootstrap, por exemplo estado de validação e controles somente

leitura. Você pode saber mais acessando a página oficial disponível na seção **Links**.

O nosso formulário para criar uma “Nova Tarefa” é o lugar perfeito para experimentar e testar os formulários no Bootstrap. Na **Listagem 21** estamos modificando o conteúdo do arquivo *app/views/tarefas/_form.html.erb* para exibir o formulário de cadastro de novas tarefas por padrão na horizontal.

Figura 8. Formulário aninhado com labels à esquerda

Figura 9. Tela de login com caixa de seleção para lembrar usuário

Figura 10. Tela de login com opção de tipo do usuário

Veja que praticamente não alteramos o conteúdo estrutural do arquivo, apenas inserimos as divs e classes CSS do Bootstrap para incutir o estilo desejado. Alguns atributos HTML inerentes às tags de input e *textarea* também podem ser inseridos livremente via Rails, como o atributo *placeholder*, que expõe uma dica dentro do campo do que deve ser digitado, e o *rows* (somente para *textarea*), que informa quantas linhas aquele campo deve exibir por padrão quando for aberto na página.

Tudo que você tem que fazer para visualizar o novo formulário é clicar em “Nova Tarefa” na página inicial do aplicativo e o mesmo o redirecionará para a página demonstrada pela **Figura 11**.

Figura 11. Tela de cadastro de tarefas com estilo do Bootstrap

Listagem 21. Exibindo formulário com Bootstrap e na horizontal.

```
01 <%= form_for(@tarefa, html: {class: 'form-horizontal'}) do |f| %>
02   <% if @tarefa.errors.any? %>
03     <div id="error_explanation">
04       <h2><%= pluralize(@tarefa.errors.count, "error") %>
05         proibiu essa tarefa de ser salva:</h2>
06     </ul>
07     <% @tarefa.errors.full_messages.each do |message| %>
08       <li><%= message %></li>
09     <% end %>
10   </ul>
11 </div>
12 <% end %>
13
14 <div class="field form-group">
15   <%= f.label :titulo, class:'col-sm-2 control-label' %><br>
16   <div class="col-sm-10">
17     <%= f.text_field :titulo, class: 'form-control', placeholder: 'Digite um título' %>
18   </div>
19 </div>
20 <div class="field form-group">
21   <%= f.label :descricao, class:'col-sm-2 control-label' %>
22   <div class="col-sm-10">
23     <%= f.text_area :descricao, class:'form-control',
24       placeholder: 'Digite uma descrição', rows: 6 %>
25   </div>
26 <div class="actions form-group">
27   <div class="col-sm-offset-2 col-sm-10">
28     <%= f.submit "Salvar" %>
29   </div>
30 </div>
31 <% end %>
```

Caso os valores do título e do botão de ‘Voltar’ na base da página estejam em inglês, basta efetuar suas respectivas traduções na página *app/views/tarefas/new.html.erb* que conecta oficialmente a página de formulário.

Navegação

Um dos principais componentes de um website ou aplicação web é o de navegação. O Bootstrap oferece as famosas *navbars*, *tabs*, *pills* e listas para que você possa decidir qual a melhor estratégia para a sua aplicação. Vamos então adicionar uma barra de navegação e alguns guias básicos. Para isso, vamos criar uma nova pasta dentro da pasta */views* chamada */util*, que conterá novos pedaços de páginas HTML que serão úteis mais à frente quando muitas páginas tiverem código em comum. Dentro dessa pasta crie um novo arquivo chamado *_navigation.html.erb* e insira o conteúdo da **Listagem 22** ao mesmo. Trata-se apenas do esqueleto da nossa navegação seguindo as recomendações do próprio Bootstrap no que se refere a *navbar* com duas ações: exibir tarefas e criar uma nova.

E em nosso arquivo *application.html.erb*, que corresponde à página principal da aplicação, precisamos renderizar o código arquétipo que acabamos de criar, conforme nos mostra a **Listagem 23**.

Rails + Bootstrap: Integrando as tecnologias na criação de aplicações web

Listagem 22. Esqueleto da nossa navbar.

```
<nav class="navbar navbar-default" role="navigation">
<div class="container">
<div class="navbar-header">
<a class="navbar-brand" href="/">Tarefas</a>
<%= link_to new_tarefa_path, class:'navbar-brand' do %>
  Nova Tarefa
<% end %>
</div>
</div>
</nav>
```

Listagem 23. Código para renderizar o esqueleto da navegação na página principal.

```
<!DOCTYPE html>
<html>
<body>
<%= render 'util/navigation'%>
</body>
</html>
```

Listagem 24. Código da tab para a index.html.erb

```
<p id="notice"><%= notice %></p>

<ul class="nav nav-tabs" role="tablist">
<li class="active"><%= link_to 'Lista de Tarefas', tarefas_path %></li>
<li><%= link_to 'Nova Tarefa', new_tarefa_path %></li>
</ul>
```

```
<h1>Lista de Tarefas</h1>
```

```
<table class="table table-bordered table-striped table-responsive">
<thead>
```

```
...
```

Listagem 25. Código da tab para a new.html.erb

```
<ul class="nav nav-tabs" role="tablist">
<li><%= link_to 'Lista de Tarefas', tarefas_path %></li>
<li class="active"><%= link_to 'Nova Tarefa', new_tarefa_path %></li>
</ul>
```

```
<h1>Nova Tarefa</h1>
```

```
<%= render 'form' %>
```

```
<%= link_to 'Voltar', tarefas_path %>
```

Feito isso, você verá na aplicação uma barra de navegação elegante exibida no navegador em sua parte superior (**Figura 12**). Não esqueça de reiniciar o servidor antes de efetuar o teste.

Tabs

As estruturas de tabs também são muito úteis para exibir partes rápidas nas páginas. Aqui, vamos adicionar o código para as nossas tabs em dois lugares: na `app/views/todos/index.html.erb` e na `app/views/todos/new.html.erb` como mostram as **Listagens 24 e 25**. Ambas as listagens inserem duas tabs na página que apontam para

as ações de listar as tarefas e criar uma nova tarefa, respectivamente. A única diferença entre os códigos nesses dois lugares é a guia que está ativa. Nós estabelecemos isso ao atribuir a classe CSS ativa (`active`) para a tab que estiver selecionada no momento.

Agora, se você visitar o app em seu navegador novamente verá que a tab “Nova Tarefa” te leva para o mesmo lugar que o botão “+ Nova Tarefa”. Também é possível voltar para o índice de todas as tarefas, clicando na aba “Lista de Tarefas”. Veja na **Figura 13** a representação final da tela.

No entanto, se você gostaria de usar os pills em vez das tabs, basta alterar a classe `nav-tabs` em cada `ul` para `nav-pills` e o efeito será automaticamente aplicado.



Figura 12. Navbar adicionada via Bootstrap no app Rails

| Lista de Tarefas | | | | |
|------------------------|------------------------|-------|--|--|
| Titulo | Descrição | Ações | | |
| fazer compras | ovos, macarrão, açúcar | | | |
| lavar o carro | | | | |
| passar com o cachorro | No mínimo 30 minutos | | | |
| fazer revisão do carro | Na oficina do Seu João | | | |
| fazer o almoço | Macarronada com fritas | | | |

Figura 13. Tela com tabs do Bootstrap

Exibição de mensagens

Quando os usuários interagem com seu aplicativo, é preciso deixá-los fornecer um feedback para saber se sua interação foi bem-sucedida ou não. O Bootstrap fornece mensagens de alerta para isso. Uma maneira comum de interagir com um alerta é usar o sistema `flash` do Rails. Isso é uma coisa simples de fazer, você simplesmente precisa percorrer as mensagens da lista do flash e exibi-las.

Vamos adicionar o seguinte código ao arquivo `app/views/layouts/application.html.erb`, o qual representa cada linha em nossa grid:

```
<% flash.each do |name, msg| -%>
<%= content_tag :div, msg, class: "alert alert-#{name}"%>
<% end -%>
```

Veja que o loop extrai duas variáveis do objeto `flash`: o `name` (classe CSS da mensagem) e a `msg` (a mensagem em si). Os alertas

Bootstrap requerem tanto uma classe de alerta, bem como uma das seguintes classes:

- *alert-success*
- *alert-info*
- *alert-warning*
- *alert-danger*

Agora nosso novo *app/views/layouts/application.html.erb* se parece com o da **Listagem 26**.

Listagem 26. Código final do arquivo de *application.html.erb*

```
<!DOCTYPE html>
<html>
<head>
<head>
<title>TarefasApp</title>
<%= stylesheet_link_tag 'application', media:'all','data-turbolinks-track' =>
  true %>
<%= javascript_include_tag 'application','data-turbolinks-track' => true %>
<%= csrf_meta_tags %>
</head>
</head>
<body>
<%= render 'util/navigation'%>
<div id="main-container" class="container">
<div class="row">
<div class="col-xs-12">
<% flash.each do |name, msg| -%>
  <%= content_tag :div, msg, class: "alert alert-#{name}" %>
<% end -%>
</div>
</div>
<div class="row">
<div class="col-xs-3">
  <h3>Aplicativo de Tarefas</h3>
</div>
<div class="col-xs-9">
  <%= yield %>
</div>
</div>
</div>
</body>
</html>
```

Após isso, precisamos mudar o nosso controlador do Rails para usar as chaves *hash* do flash adequadas para trabalhar com os nossos alertas. Vamos começar com a mensagem de sucesso quando a tarefa é criada.

Nossas mudanças devem ser feitas diretamente no arquivo *app/controllers/tarefas_controller.rb*. Portanto, busque pelas três linhas a seguir no referido arquivo:

```
format.html { redirect_to @tarefa, notice: 'Tarefa was successfully created.'}
format.html { redirect_to @tarefa, notice: 'Tarefa was successfully updated.'}
format.html { redirect_to tarefas_url, notice: 'Tarefa was successfully destroyed.'}
```

Essas são as linhas que precisamos de fato atualizar. Atualmente, o nosso aplicativo está usando a chave de *hash* chamada *notice* no *hash* do flash (que é armazenada na sessão, por sinal), isto é, no objeto que guarda todas as mensagens da aplicação. Essa chave é necessária para que o Rails saiba exatamente de onde estão vindo tais mensagens, além de incutir seus procedimentos internos de segurança. O Rails fornece duas chaves de flash para exibir mensagens por padrão: *notice* e *error*, mas uma vez que for preciso usar uma chave de sucesso, teremos de mover nossa mensagem flash para fora do método *redirect_to*. Vejamos então como proceder em alguns passos básicos.

- **Código antigo:**

```
if @tarefa.save
  format.html { redirect_to @tarefa, notice: 'Tarefa was successfully created.'}
```

- **Código novo:**

```
if @tarefa.save
  flash[:success] = 'Tarefa foi criada com sucesso.'
  format.html { redirect_to @tarefa }
```

O mesmo procedimento deve ser tomado para os demais exemplos de edição e remoção. Agora se você criar, editar ou remover uma tarefa, verá um alerta verde no topo da página informando que tal ação foi efetuada com sucesso, tal como demonstrado na **Figura 14**.



Figura 14. Exemplo de mensagem de validação após inserção

Alertas de falha

Precisamos exibir um erro se houver uma falha no nosso controlador. Semelhante aos alertas de sucesso, precisamos adicionar mensagens de *danger* (a classe Bootstrap para alertas vermelhos) à chave do flash no Rails.

Nas **Listagens 27** e **28** estão as ações finais do controlador.

Agora você tem um mecanismo simples e agradável para notificar os usuários sempre que algo acontecer na sua aplicação. Não se esqueça de alterar o *success* e o *danger* para uma das classes mencionadas afim de garantir a mudança de cor do alerta via Bootstrap. É importante sempre mostrar ao usuário o que acontece na aplicação para que o mesmo entenda o fluxo das coisas e não ache que algo de errado aconteceu.

Modals

Os *modals* são espécies de pop-ups que estão contidas dentro da sua página processada e não necessitam abrir outra janela ou guia do navegador para exibi-las.

Listagem 27. Definição final da função de criação das tarefas, com mensagem de erro.

```
def create
  @tarefa = Tarefa.new(tarefa_params)

  respond_to do |format|
    if @tarefa.save
      flash[:success] = 'Tarefa foi criada com sucesso.'
      format.html { redirect_to @tarefa }
      format.json { render :show, status: :created, location: @tarefa }
    else
      flash[:danger] = 'Houve um problema ao criar a Tarefa.'
      format.html { render :new }
      format.json { render json: @tarefa.errors, status: :unprocessable_entity }
    end
  end
end
```

Listagem 28. Definição final da função de edição das tarefas, com mensagem de erro.

```
def update
  respond_to do |format|
    if @todo.update(todo_params)
      flash[:success] = 'Todo was successfully updated.'
      format.html { redirect_to @todo }
      format.json { render :show, status: :ok, location: @todo }
    else
      flash[:danger] = 'There was a problem updating the Todo.'
      format.html { render :edit }
      format.json { render json: @todo.errors, status: :unprocessable_entity }
    end
  end
end
```

Para a nossa aplicação, vamos usar as *modals* para exibir as descrições das nossas tarefas. Para isso, precisamos fazer essas mudanças diretamente no arquivo *app/views/tarefas/index.html.erb*. Em vez de exibir as descrições na tabela vamos mostrar um link que irá ativar a referida modal expondo o próprio texto nela. Se a tarefa não tem descrição então não haverá link sendo exibido. Precisamos mudar, portanto, o código da segunda *td* na tabela (*<td><%= todo.notes %></td>*) para o exibido na **Listagem 29**. Veja que logo no início testamos se existe uma descrição ou não para, caso não haja, nem procedermos com a execução do código. A estrutura de um modal no Bootstrap é muito simples: basta criar um gatilho (no caso um botão ou link) que irá disparar a abertura do mesmo. O mesmo botão/link precisa ter dois atributos do Bootstrap:

- *data-toggle*: para informar que o botão abrirá um elemento do tipo “modal” após seu click;
- *data-target*: informa qual o id do modal que deve ser aberto, já que podemos ter vários na tela. Inclusive essa declaração toda está sendo feita dentro do *foreach* justamente para que possamos criar um modal distinto para cada linha da tabela. Garantimos essa unicidade de identificador através do próprio id do objeto na base de dados.

O restante da implementação traz apenas a montagem da div de modal que deve estar anotada com as classes CSS do Bootstrap, além de conter os atributos:

- *role*: define o tipo de regra ao qual o elemento se aplica, no caso um modal;
- *aria-label*: o texto que deve aparecer como título do modal;
- *aria-hidden*: booleano para informar se o modal deve iniciar escondido ou não.

Para finalizar, o botão de fechamento deve conter o atributo *data-dismiss* para indicar que o seu click irá fechar a janela do modal.

Quando você atualizar o seu navegador, verá que sua tabela tem agora mais um elemento na coluna de “Descrição”. Ao clicar no mesmo a janela demonstrada pela **Figura 15** será automaticamente aberta.

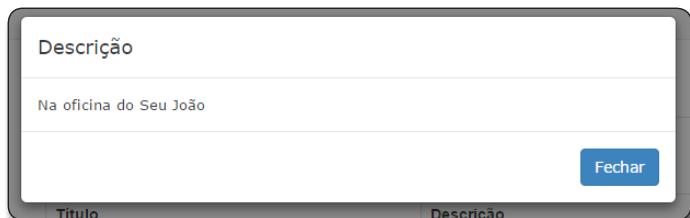


Figura 15. Exemplo de modal aberta em função do click no novo botão

Listagem 29. Código da td de descrição com modal.

```
<td>
<% if tarefa.descricao? %>
<button class="btn btn-link" data-toggle="modal" data-target="#tarefa-<%= tarefa.id %>-descricao">
  Ver Descrição
</button>

<!-- Modal -->
<div class="modal fade" id="tarefa-<%= tarefa.id %>-descricao" tabindex="-1"
  role="dialog" aria-labelledby="tarefa-<%= tarefa.id %>-label"
  aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h4 class="modal-title" id="tarefa-<%= tarefa.id %>-label">
          Descrição</h4>
      </div>
      <div class="modal-body">
        <%= tarefa.descricao %>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-primary"
          data-dismiss="modal">Fechar</button>
      </div>
    </div>
  </div>
<% end %>
</td>
```

Essas foram apenas algumas das funcionalidades mais usuais e importantes de ambos os frameworks que podemos fazer uso no dia a dia como desenvolvedores. Tais situações implicam nos famosos CRUDs que são muito mais comuns e abrangem um cenário mais globalizado quando se trata de aplicações web como um todo. O grande segredo do Bootstrap é saber como funcionam suas bases e analisar a melhor estratégia para acoplá-lo à plataforma server side.

Tome sempre bastante cuidado com outras bibliotecas JavaScript ou CSS inclusas no seu projeto, pois isso pode causar problemas de compatibilidade, principalmente se as mesmas forem pertencentes a projetos semelhantes, como o Material Design ou o AngularJS, por exemplo.

Autor



Fabrício Galdino

É um especialista em software e trabalhou com análise de TI e desenvolvimento de negócios por cinco anos. Tem experiência com testes e tecnologias relacionadas ao front-end, como SEO, Responsividade, HTML5 e CSS3.



Links:

Página de download do RubyInstaller for Windows

<http://rubyinstaller.org/>

Página oficial do LESS

<http://lesscss.org/>

Página oficial do Bootstrap

<http://getbootstrap.com.br/>

Desenvolvimento de jogos web com Pixi.js – Parte 3

Crie jogos e animações gráficas para a web usando a API JavaScript do Pixi.js com HTML5

ESTE ARTIGO FAZ PARTE DE UM CURSO

O Pixi.js é um framework muito robusto que atende à grande maioria das implementações que você venha a precisar no desenvolvimento dos seus jogos ou aplicações gráficas na web. Entretanto, muitos dos recursos precisam fazer uso de APIs de terceiros e/ou bibliotecas utilitárias para atender a implementações não fornecidas pelo mesmo, como o mecanismo de detecção de colisões entre objetos nos jogos. Esse tipo de funcionalidade é extremamente importante, principalmente em jogos clássicos que seguem o estilo de personagem principal contra personagens inimigos. Além disso, mesmo que não tenhamos o choque entre personagens ao longo do jogo, o mínimo de detecção de espaço entre os limites do cenário e o personagem nele inserido precisa ser atendido pelo seu projeto.

E é justamente sobre isso que iremos tratar neste artigo: como detectar a colisão entre dois objetos e, a partir disso, determinar o que acontecerá com os mesmos e com o cenário como um todo. É importante salientar que esse recurso precisa ser o mais genérico possível, retornando sempre um booleano e recebendo os objetos que necessitam ser checados quanto à colisão; uma vez que só determinaremos a ação pós-colisão quando a mesma ocorrer. O leitor precisa, a este ponto, entender que os conceitos aprendidos até aqui se intercalam e comunicam entre si, tendo dependência direta e hierárquica. Por exemplo, para que possamos criar uma

Fique por dentro

Este artigo é útil por explorar os principais conceitos na prática acerca da biblioteca de desenvolvimento de gráficos 2D Pixi.js. Essa biblioteca ficou famosa por sua capacidade de abstrair o WebGL (padrão baseado no OpenGL ES 2.0 que fornece interfaces de programação de gráficos em 3D) de forma rápida e extremamente leve em comparação com outras ferramentas do tipo. O leitor aprenderá a implementar um mecanismo próprio e robusto de detecção de colisões entre objetos no jogo e, sobretudo, a como executar ações quando elas ocorrerem. Esse trabalho se dará através da biblioteca utilitária do Bump.js, que fornece uma linha de execução pronta para detectar as colisões. Veremos tudo isso através da construção de um jogo simples de caça à tocha em um cenário repleto de personagens inimigos, com barra de vida e cenas de game over e sucesso.

função que detecte uma colisão, precisamos antes criar os objetos de sprite via Texture Packer e JSON, bem como mapear seus eventos de clique nas teclas que resultarão na sua movimentação básica, etc. Portanto, uma boa revisão nos conceitos vistos seria muito aconselhada.

Detectando colisões

Agora que já temos a estrutura básica necessária para trabalhar, em JSON, com a criação dinâmica de sprites adicionando-os ao cenário, podemos partir para uma técnica muito importante no universo dos games: a detecção de colisões entre os objetos. Até o momento, nossos objetos apenas desempenham tarefas simples, como a caminhada (ou corrida, na qual incluímos os conceitos de velocidade, aceleração e fricção) e os eventos de movimento associados às teclas das setas do seu teclado.

O Pixi.js não fornece, por padrão, um mecanismo nativo de detecção de colisões, mas podemos fazer uso da biblioteca chamada Bump.js (vide seção **Links**), que já se integra automaticamente ao Pixi, disponibilizando uma série de funções para lidar com a manipulação de colisões em jogos 2D.

Como primeiro passo, vamos efetuar o download da biblioteca (apenas do arquivo bump.js) e importar o mesmo na nossa página HTML logo abaixo do script do Pixi (é interessante considerar a criação de um novo arquivo para o código que faremos a seguir). Finalmente, para fazer uso da API, basta criar um novo objeto do tipo *Bump* na sua função setup(), tal como:

```
var bump = new Bump(PIXI);
```

Veja que também passamos o objeto *PIXI* como parâmetro, já que a mesma precisa receber a engine de jogos que estamos usando.

O que vamos usar da API em si basicamente é o seu método principal: a função hitTestRectangle() (acesse o arquivo JavaScript para mais detalhes da sua implementação). Essa função, por sua vez, recebe dois parâmetros: os dois sprites a serem comparados; e retorna um booleano informando se há alguma colisão entre ambos. Assim, podemos usá-la em estruturas condicionais para atestar se houve ou não alguma colisão naquele momento. Na maioria dos casos, a checagem de colisão deve ser feita em todos os frames do jogo, portanto é necessário incluir tal lógica dentro da função play() que será chamada a cada FPM.

Prática de jogo: Em busca da tocha!

Com o intuito de assimilar a ideia da detecção de colisão e unir a mesma aos conceitos vistos até aqui, faremos um jogo completo e bem simples, cujo nome será “Em busca da tocha!”. Teremos um player central, um garoto, que terá por objetivo buscar uma tocha dentro do cenário que estará cercado por inimigos representados pelos personagens de fantasmas. Também exibiremos uma barra de vida no topo do cenário na cor verde, que decrescerá a medida que o personagem principal for atingido por algum dos fantasmas. Se ele for atingido o suficiente para zerar a barra, o jogo encerrará e o jogador será redirecionado para a tela de game over. Caso contrário, ele terá de levar a tocha até a porta de saída do cenário, no canto superior esquerdo, aí sim o jogador será bem-sucedido e ganhará o jogo e então poderemos direcioná-lo para a tela de sucesso.

Também implementaremos a detecção de colisão com as bordas do cenário, impossibilitando o jogador de ir além das paredes. E, para completar o jogo, criaremos um texto com o título do jogo e um contador regressivo antes do jogo começar para preparar o jogador, limpando os referidos objetos assim que o contador chegar em zero.

Para iniciar, crie um novo arquivo HTML (chamaremos aqui de cacatocha.html) e inicialize-o com a estrutura de HTML e imports demonstrados na **Listagem 1**. A mesma já nos é conhecida dos exemplos anteriores.

Antes de começar a criação do conteúdo JavaScript da página, precisamos agrupar os arquivos de imagens que usaremos como sprites para o jogo no Texture Packer. Para isso, efetue o download do código fonte deste artigo e importe as imagens do cenário, da porta, da tocha e dos personagens no referido software, efetuando o mesmo processo de exportação do arquivo PNG final, bem como do JSON de mapeamento (veja na **Figura 1** o resultado da geração). Daremos os nomes de cacaTacha.png e cacaTacha.json, respectivamente.

Listagem 1. Estrutura inicial da página cacatocha.html.

```
01 <!doctype html>
02 <meta charset="utf-8">
03 <title>Caça à Toca!</title>
04 <style type='text/css'>
05   *{
06     padding: 0;
07     margin: 0
08   }
09
10  @font-face {
11    font-family: 'Pipe Dream';
12    src: url('font/Pipe_Dream.ttf') format('truetype');
13    font-weight: normal;
14    font-style: normal;
15  }
16 </style>
17
18 <body>
19   <script src="js/pixi.min.js"></script>
20   <script>
21
22   </script>
23   <div style="font-family: Pipe Dream;">.</div>
24 </body>
25
26 </html>
```



Figura 1. Sprites do jogo de Caça à Toca!

Após isso, dentro da tag <script> na nossa página, insira o código da **Listagem 2**. Veja que até aqui não apresentamos ainda novidades, exceto pela importação do arquivo JSON que traz novos objetos. Para facilitar o trabalho de uso dos objetos, criaremos alias para cada um dos objetos de sprite que manipularemos ao longo do código (linhas 2 a 10). Em jogos reais, você deve sempre procurar fazer uso desse recurso, já que a tendência é que a quantidade de linhas seja muito grande. Nas linhas 14 a 16 inicializamos os objetos de *stage* e *renderer*, definindo dimensões base de 512x512 para a tela do jogo (lembre-se que tais dimensões devem corresponder às mesmas das imagens de plano de fundo dos cenários).

Listagem 2. Código de inicialização dos objetos de alias, container e função de setup().

```
01 // Alias
02 var Container = PIXI.Container,
03     autoDetectRenderer = PIXI.autoDetectRenderer,
04     loader = PIXI.loader,
05     resources = PIXI.loader.resources,
06     TextureCache = PIXI.utils.TextureCache,
07     Texture = PIXI.Texture,
08     Sprite = PIXI.Sprite,
09     Text = PIXI.Text,
10    Graphics = PIXI.Graphics;
11
12 // Cria um stage e um renderer do Pixi e adiciona
13 // o renderer.view ao DOM
14 var stage = new Container(),
15     renderer = autoDetectRenderer(512, 512);
16 document.body.appendChild(renderer.view);
17
18 loader
19   .add("img/cacatocha/cacaTacha.json")
20   .load(setup);
21
22 // Define as variáveis que devem ser usadas em mais
23 // de uma função
24 var state, garoto, tocha, fantasmas, player, cenario,
25     porta, barraSaude, mensagem, gameScene, gameOverScene, inimigos, id;
26
27 function setup() {
28
29 }
```

Na linha 18 inicializamos o objeto de *loader*, passando para o mesmo o caminho do arquivo JSON de mapeamento dos tilesets, e na linha 24 criamos todas as referências dos objetos que precisaremos no restante do código. Finalmente, na linha 27 declaramos a função de configuração *setup*.

Na **Listagem 3** você encontra o código inicial da função de *setup()*. Como temos muitas coisas para configurar na mesma, a dividiremos em duas partes para facilitar o entendimento. As mesmas estão comentadas, mas vamos considerar alguns pontos importantes:

- Usaremos o método *addChild()* do *stage* sempre que quisermos adicionar um elemento ao cenário (linhas 3, 10, 15, 23, 29 e 66). O mesmo objeto também conta com um método *removeChild()* para efetuar o processo inverso.

- Na linha 6 recuperamos os objetos de texturas associadas ao arquivo JSON. Veja que adicionamos o vetor à variável *id*, mas você pode ficar à vontade para usar o nome que desejar.

- Na linha 14 apresentamos a função *set()* dos objetos de sprites. Cada sprite, como vimos, contém um elemento de *position* interno referente às duas posições no vetor de plano cartesiano (x e y). É possível configurar tais valores manualmente, como fazemos mais abaixo no código do sprite de garoto, usando as propriedades brutais *x* e *y*. A sequência restante do código trata exclusivamente da criação e adição dos elementos do jogo na tela: cenário, porta, garoto, tocha e os fantasmas.

- Veja que na linha 20 estamos calculando o valor da posição vertical (y) do player do garoto subtraindo ambas as divisões da altura da cena e da altura do objeto garoto, respectivamente. Isso nos dará exatamente o valor central onde o mesmo deverá aparecer quando o jogo iniciar, e você deve usar sempre esse tipo de código quando quiser posicionar elementos relativamente aos demais na tela. Não confie em projeções de valores fixos, pois o mesmo jogo poderá ser executado em diferentes tamanhos e resoluções de telas.

- Já para o objeto de tocha (linhas 27 e 28), modificamos a sua localização para o mais próximo da base do jogo, dando um *padding* (48 pixels) para que ela não toque as paredes do cenário.

- Quanto aos objetos de fantasmas, tivemos a preocupação de configurar um código, no mínimo, autoajustável. Isso porque dependendo do desenvolvedor, dos interesses do seu jogo ou, inclusive, da necessidade de fases mais difíceis no decorrer do mesmo, uma quantidade maior de fantasmas pode ser necessária. As variáveis que precisaremos manipular para atingir tal objetivo, portanto, seriam:

- *numDeFantasmas*: configura a quantidade de fantasmas que aparecerão na tela do jogo;
- *spacing*: determina o espaço em pixels entre cada um dos objetos;
- *xOffset*: configura a distância que o primeiro objeto de fantasma deve ter da extremidade esquerda da tela;
- *speed*: velocidade dos objetos;
- *direction*: direção. Usaremos o valor “1” para movê-lo para baixo, e “-1” para cima.

- No fim, basta criar um loop (*forEach*) de acordo com a quantidade de fantasmas que serão adicionados, posicionando-os randomicamente na tela. Para essa finalidade, veja que fizemos uso da função *randomInt()* do JavaScript (linha 50), que atuará calculando o valor y de cada objeto aleatoriamente entre 0 e a largura máxima da tela subtraída da altura do próprio objeto. Para lidar com a direção do objeto, a grande sacada é fazer uso do jogo de sinais entre os valores 1 e -1, isso porque ao multiplicar um com o outro teremos sempre o valor inverso, com sinal inverso, e, portanto, saberemos para onde o objeto deve seguir.

Na sequência, temos o conteúdo da **Listagem 4**, que complementa a anterior no que remete à função *setup()*.

Aqui, tratamos de criar e adicionar os demais objetos seguindo a mesma lógica de posicionamento, lidando com a barra de

Listagem 3. Código de configuração dos objetos, setup() – Parte 1.

```
01 // Constrói a cena do jogo e a adiciona ao stage
02 gameScene = new Container();
03 stage.addChild(gameScene);
04
05 // Cria os sprites e os add ao `gameScene`
06 id = resources["img/cacatocha/cacaTacha.json"].textures;
07
08 // Cenário
09 cenario = new Sprite(id["cenario.png"]);
10 gameScene.addChild(cenario);
11
12 // Porta
13 porta = new Sprite(id["porta.png"]);
14 porta.position.set(32, 0);
15 gameScene.addChild(porta);
16
17 // Garoto explorador
18 garoto = new Sprite(id["garoto.png"]);
19 garoto.x = 68;
20 garoto.y = gameScene.height/2 - garoto.height/2;
21 garoto.vx = 0;
22 garoto.vy = 0;
23 gameScene.addChild(garoto);
24
25 // Tacha
26 tocha = new Sprite(id["tocha.png"]);
27 tocha.x = gameScene.width - tocha.width - 48;
28 tocha.y = gameScene.height - tocha.height - 48;
29 gameScene.addChild(tocha);
30
31 // Cria os fantasmas
32 var numDeFantasmas = 6,
33    spacing = 48,
34    xOffset = 150,
35    speed = 2,
36    direction = 1;

37
38 // Um vetor para armazenar todos os fantasmas
39 fantasmas = [];
40
41 // Cria quantos fantasmas a variável `numDeFantasmas` informar
42 for (var i = 0; i < numDeFantasmas; i++) {
43   // Cria um fantasma
44   var fantasma = new Sprite(id["fantasma.png"]);
45
46   // Dá um espaço para cada fantasma horizontalmente de acordo com o valor do
   `spacing`. `xOffset` determina o ponto a partir da esquerda da tela em que o
   primeiro fantasma deve ser adicionado
47   var x = spacing * i + xOffset;
48
49   // Dá ao fantasma uma posição randômica y
50   var y = randomInt(0, stage.height - fantasma.height);
51
52   // Configura a posição do fantasma
53   fantasma.x = x;
54   fantasma.y = y;
55
56   // Configura a velocidade vertical do fantasma. `direction` será `1` ou `-1`.
   `1` significa que o inimigo se moverá para baixo e `-1` o contrário.
   Ao multiplicar a `direction` pela `speed` determinamos a direção vertical do
   fantasma
57   fantasma.vy = speed * direction;
58
59   // Reverte a direção do próximo fantasma
60   direction *= -1;
61
62   // Insere o fantasma no vetor de `fantasmas`
63   fantasmas.push(fantasma);
64
65   // Add o fantasma ao `gameScene`
66   gameScene.addChild(fantasma);
67 }
```

saúde e as cenas de entrada e finalização do jogo. Vejamos alguns detalhes:

- O objeto de *barraSaude* será composto de outros dois objetos: a barra de saúde restante (em verde) e barra de saúde perdida (em vermelho). Por essa razão, o objeto em si deve ser do tipo *Container*, um objeto invisível do Pixi que permite a inclusão de subelementos no mesmo e serve apenas para a finalidade de “conter” os demais. Sua posição será na parte superior esquerda da tela.
- Para criar ambas as barras precisaremos desenhar um retângulo, literalmente, na tela. O Pixi fornece um mecanismo de gráficos muito semelhante ao de outras linguagens (como o Graphics2D do Java, por exemplo) para lidar com a impressão de objetos geométricos. Tudo que precisamos fazer é instanciar um objeto do tipo *Graphics* (linhas 7 e 14), inicializar suas cores via função *beginFill()* (atente que as cores devem ser passadas em seu formato hexadecimal), desenhar as dimensões do retângulo via função *drawRect()* (neste caso ambas as barras terão a mesma altura – 8 pixels – e mesma largura – 128 pixels) e fechar o objeto chamando a função *endFill()*. Depois de criá-las e adicioná-las ao stage, podemos definir a propriedade *outer* do objeto de container (linha 20), que dirá qual objeto aparecerá preenchido no container.

- O restante das configurações, incluindo a cena de game over, já nos são bem conhecidas, exceto pela função *carregarTexto()* que vemos impressa no fim do código. Veremos seu conteúdo mais à frente.

Em sequência, temos na **Listagem 5** o código referente à configuração dos eventos de ouvintes das teclas de seta para movimentação do personagem principal. O motivo de ainda não termos efetuado a chamada dessa função na função de *setup()* é justamente por precisarmos que o personagem não tenha nenhum movimento quando o jogo iniciar, mas sim apenas quando o contador regressivo de que falamos finalizar. Assim, a funcionalidade encapsulada nessa função poderá ser executada de qualquer parte do código.

Veja que as configurações são praticamente iguais às que fizemos antes no artigo anterior, portanto não nos ateremos a muitos detalhes. A função *gameLoop()* da linha 54 inicializa o processo de renderização do jogo.

Após isso, precisamos garantir a inclusão do contador regressivo que inicializará o jogo de fato, dando tempo suficiente para o jogador se preparar para o mesmo. Temos duas funções que lidarão com isso, a saber na **Listagem 6**:

Desenvolvimento de jogos web com Pixi.js – Parte 3

- Função carregarTexto(): se encarrega de criar o objeto de texto inicial (“Em busca da Tocha!”), definindo as propriedades de fonte, tamanho, sombra, etc. Posicionaremos a mensagem um pouco abaixo do topo da tela e a adicionaremos ao container logo quando o jogo iniciar. Para a contagem regressiva faremos uso de recursividade em conjunto com a função setTimeout()

do JavaScript para estabelecer os intervalos de tempo entre uma chamada e outra. Para tanto, precisaremos de uma variável que será decrementada a partir do valor 3 (linha 16).

- Função txtContadorRecursivo(): recebe o valor do contador que será usado para imprimir um segundo objeto de texto, apenas com esse valor. Veja que configuramos no final da função um segundo

Listagem 4. Código de configuração dos objetos, setup() – Parte 2.

```
01 // Cria a barra de saúde
02 barraSaude = new Container();
03 barraSaude.position.set(stage.width - 170, 6)
04 gameScene.addChild(barraSaude);
05
06 // Cria o retângulo de fundo vermelho
07 var innerBar = new Graphics();
08 innerBar.beginFill(0xFF3300);
09 innerBar.drawRect(0, 0, 128, 8);
10 innerBar.endFill();
11 barraSaude.addChild(innerBar);
12
13 // Cria o retângulo da frente verde
14 var outerBar = new Graphics();
15 outerBar.beginFill(0x8BC34A);
16 outerBar.drawRect(0, 0, 128, 8);
17 outerBar.endFill();
18 barraSaude.addChild(outerBar);
19
20 barraSaude.outer = outerBar;
21
22 // Cria a cena de `gameOver`
23 gameOverScene = new Container();
24 stage.addChild(gameOverScene);
```



```
25
26 // Faz a cena de `gameOver` ficar invisível quando o jogo iniciar
27 gameOverScene.visible = false;
28
29 // Cria o sprite de texto e o adiciona à cena de `gameOver`
30 mensagem = new Text(
31   "Fim!",
32   {font:"64px Pipe Dream",
33    fill:"white"
34   }
35 );
36 mensagem.x = 120;
37 mensagem.y = stage.height/2 - 32;
38 gameOverScene.addChild(mensagem);
39
40 // Configura o estado do jogo
41 state = play;
42
43 // Inicia o game loop
44 gameLoop();
45
46 // Carrega o texto de contagem regressiva
47 carregarTexto();
```

Listagem 5. Configuração dos eventos de teclas.

```
01 function configKeys() {
02   // Captura as keys de seta do teclado
03   var left = keyboard(37),
04     up = keyboard(38),
05     right = keyboard(39),
06     down = keyboard(40);
07
08   left.press = function() {
09     // Muda a velocidade do garoto quando a tecla é pressionada
10     garoto.vx = -5;
11     garoto.vy = 0;
12   };
13
14   left.release = function() {
15     // Se a tecla de seta esquerda for solta, e a seta direita não estiver pressionada,
16     // e o garoto não estiver se movendo na vertical:
17     // Para o garoto
18     if (!right.isDown && garoto.vy === 0) {
19       garoto.vx = 0;
20     }
21   };
22
23   up.press = function() {
24     garoto.vy = -5;
25     garoto.vx = 0;
26   };
27   up.release = function() {
28     if (!down.isDown && garoto.vx === 0) {
29       garoto.vy = 0;
30     }
31   };
32
33   right.press = function() {
34     garoto.vx = 5;
35     garoto.vy = 0;
36   };
37   right.release = function() {
38     if (!left.isDown && garoto.vy === 0) {
39       garoto.vx = 0;
40     }
41   };
42
43   down.press = function() {
44     garoto.vy = 5;
45     garoto.vx = 0;
46   };
47   down.release = function() {
48     if (!up.isDown && garoto.vx === 0) {
49       garoto.vy = 0;
50     }
51   };
52 }
53
54 function gameLoop() {
55   requestAnimationFrame(gameLoop);
56
57   state();
58
59   renderer.render(stage);
60 }
```

timeout para remover o mesmo objeto da tela assim que se passar exatamente um segundo. Assim, garantimos que todos os objetos estarão completamente removidos após os três segundos de contagem. Veja que, na função anterior, também inserimos um quarto timeout para remover a mensagem com o nome do jogo, além de chamar a função configKeys() que, assim, liberará os movimentos do player uma vez que o contador tenha chegado ao fim.

A **Listagem 7**, por sua vez, exibe as funções de início e fim do jogo. É aqui onde inicia nosso processo de checagem de colisões, de fato. Vejamos alguns detalhes:

- Nas duas primeiras linhas tratamos de mover o objeto do garoto ao longo da tela quando o respectivo click nas teclas de setas for efetuado.
- Na linha 7 fazemos uma checagem para verificar se o garoto está dentro do espaço da cena ou não via função contain(). Essa função se encarregará de sempre checar isso, nos auxiliando no processo de limitação do cenário mediante os personagens. Veremos seu código mais à frente.
- Na linha 18 estamos iterando sobre cada um dos objetos de fantasmas usando a função forEach do JavaScript (uma alternativa à each() do jQuery). No loop, movemos cada um dos objetos tal qual fizemos com o garoto e calculamos se o mesmo está em contato com algum outro objeto (linha 23), configurando na variável fantasmaColideParede tal valor. Nas linhas seguintes checamos a colisão para inverter seu movimento através do jogo de sinais que já vimos. A função hitTestRectangle(), cujo código também veremos adiante, verifica se ambos os objetos estão em colisão, caso positivo configuramos o valor da variável de garotoColide para true.
- Na linha 43 testamos se houve alguma colisão do garoto com qualquer um dos fantasmas, caso positivo devemos demonstrar

para o usuário que isso está acontecendo através da mudança na opacidade do objeto via código HTML. A propriedade alpha dos objetos Pixi possibilita incutir exatamente esse comportamento, sobrescrevendo a propriedade *opacity* do CSS (para essa propriedade, o valor deve estar entre 0 e 1 – onde 0 representa total invisibilidade e 1 representa total exibição do objeto). Além disso, quando o personagem for atingido, devemos decrescer o tamanho da barra verde de vida e faremos isso através da propriedade *width* do referido objeto de *Rectangle*. Caso não tenha ocorrido nenhuma colisão, o objeto volta para o estado de exibição completa.

- Na linha 55 verificamos mais uma vez se uma colisão ocorreu para que possamos centralizar o objeto de tocha junto ao objeto do garoto. Tais valores podem ser customizados para exibir a tocha à esquerda ou em qualquer outra posição que desejar.
- Na próxima verificação da linha 63 precisamos verificar se a barra de vida do usuário chegou ao fim. Caso seu tamanho seja menor que zero, então nosso personagem foi suficientemente atingido e perdeu. Portanto, mudamos o estado do jogo para finalizado (*end*) e imprimimos no texto da mensagem o novo valor de “Perdeu!”.
- Caso contrário, se o jogador tiver atingido o objeto de porta, finalizamos o jogo, porém exibindo a nova mensagem de vitória ao usuário.
- Por fim, a função *end()* no fim da listagem apenas esconde a cena comum do jogo e exibe a cena de game over configurando o atributo *visible* das mesmas.

Dando sequência, na **Listagem 8** temos algumas funções utilitárias para lidar com a detecção de colisão entre os objetos envolvidos, bem como a função que “ouve” os eventos de teclado para movimento do personagem principal. Vejamos alguns detalhes das mesmas:

Listagem 6. Função de contador regressivo.

```

01 function carregarTexto() {
02     var helloMsg = new PIXI.Text(
03         "Em Busca da Tocha!", {
04             font: "38px Pipe Dream",
05             fill: "white",
06             dropShadow: true,
07             dropShadowDistance: 2,
08             dropShadowColor: "black"
09         }
10     );
11     helloMsg.x = renderer.view.width/2 - helloMsg.width/2;
12     helloMsg.y = helloMsg.height + 78;
13
14     stage.addChild(helloMsg);
15
16     var cont = 3;
17     txtContadorRecursivo(cont--);
18     setTimeout(function() {
19         txtContadorRecursivo(cont--);
20         setTimeout(function() {
21             txtContadorRecursivo(cont--);
22             setTimeout(function() {
23                 stage.removeChild(helloMsg);
24                 configKeys();
25             }, 1000);
26         }, 1000);
27     }, 1000);
28 }
29
30 function txtContadorRecursivo(cont) {
31     var contMsg = new PIXI.Text(
32         cont, {
33             font: "38px Pipe Dream",
34             fill: "white",
35             dropShadow: true,
36             dropShadowDistance: 2,
37             dropShadowColor: "black"
38         }
39     );
40     contMsg.x = renderer.view.width/2 - contMsg.width/2;
41     contMsg.y = contMsg.height + 118;
42
43     stage.addChild(contMsg);
44
45     setTimeout(function() {
46         stage.removeChild(contMsg);
47     }, 1000);
48 }
```

Desenvolvimento de jogos web com Pixi.js – Parte 3

Listagem 7. Funções de play() e end() para iniciar e finalizar o jogo.

```
01 function play() {
02   // Usa a velocidade do garoto para movê-lo
03   garoto.x += garoto.vx;
04   garoto.y += garoto.vy;
05
06   // Checa se o garoto está dentro da área do cenário
07   contain(garoto, {
08     x: 12,
09     y: 10,
10     width: 498,
11     height: 480
12   });
13
14   // Configura o `garotoColide` para `false` antes de checar se houve colisão
15   var garotoColide = false;
16
17   // Faz um loop ao longo de todos os sprites no vetor de `inimigos`
18   fantasmas.forEach(function(fantasma) {
19     // Move o fantasma
20     fantasma.y += fantasma.vy;
21
22     // Checa os limites da tela do fantasma
23     var fantasmaColideParede = contain(fantasma, {
24       x: 28,
25       y: 32,
26       width: 488,
27       height: 480
28     });
29
30     // Se o fantasma atingir o top ou bottom do stage, reverta sua direção
31     if (fantasmaColideParede === "top" || fantasmaColideParede === "bottom") {
32       fantasma.vy *= -1;
33     }
34
35     // Testa se houve uma colisão. Se qualquer um dos inimigos estiver tocando o
36     // garoto, configura o `garotoColide` para `true`
37     if (hitTestRectangle(garoto, fantasma)) {
38       garotoColide = true;
39     }
40   });
41
42   // Se o garoto for atingido...
43   if (garotoColide) {
44     // Faz o garoto semitransparente
45     garoto.alpha = 0.5;
46
47     // Reduz a largura da barra de saúde em 1 pixel
48     barraSaude.outer.width -= 1;
49   } else {
50     // Faz que o garoto fique totalmente opaco (não transparente)
51     // se ele não tiver sido atingido
52     garoto.alpha = 1;
53
54     // Checa se há uma colisão entre o garoto e a tocha
55     if (hitTestRectangle(garoto, tocha)) {
56       // Se a tocha estiver estiver tocando o garoto, a centraliza sobre o garoto
57       tocha.x = garoto.x + 20;
58       tocha.y = garoto.y + 6;
59     }
60
61     // O garoto tem vida suficiente? Se a largura da `innerBar`
62     // é menor que zero, finaliza o jogo e exibe a mensagem "Perdeu!"
63     if (barraSaude.outer.width < 0) {
64       state = end;
65       mensagem.text = "Perdeu!";
66     }
67
68     // Se o garoto trouxer a tocha para a saída,
69     // finaliza o jogo e exibe a mensagem "Victory!"
70     if (hitTestRectangle(tocha, porta)) {
71       state = end;
72       mensagem.text = "Victory!";
73     }
74
75
76   function end() {
77     gameScene.visible = false;
78     gameOverScene.visible = true;
79 }
```

Listagem 8. Funções utilitárias para detecção de colisão.

```
01 function contain(sprite, container) {
02   var collision = undefined;
03   if (sprite.x < container.x) {
04     sprite.x = container.x;
05     collision = "left";
06   }
07   if (sprite.y < container.y) {
08     sprite.y = container.y;
09     collision = "top";
10   }
11   if (sprite.x + sprite.width > container.width) {
12     sprite.x = container.width - sprite.width;
13     collision = "right";
14   }
15   if (sprite.y + sprite.height > container.height) {
16     sprite.y = container.height - sprite.height;
17     collision = "bottom";
18   }
19   return collision;
20 }
21
22 function hitTestRectangle(r1, r2) {
23   var hit, combinedHalfWidths, combinedHalfHeights, vx, vy;
```

```
24   // Determinará quando há ou não colisão
25   hit = false;
26   // Encontra os pontos centrais de cada sprite
27   r1.centerX = r1.x + r1.width/2;
28   r1.centerY = r1.y + r1.height/2;
29   r2.centerX = r2.x + r2.width/2;
30   r2.centerY = r2.y + r2.height/2;
31
32   // Encontra a metade da largura e altura de cada sprite
33   r1.halfWidth = r1.width/2;
34   r1.halfHeight = r1.height/2;
35   r2.halfWidth = r2.width/2;
36   r2.halfHeight = r2.height/2;
37
38   // Calcula o vetor de distância entre os sprites
39   vx = r1.centerX - r2.centerX;
40   vy = r1.centerY - r2.centerY;
41
42   // Descobre quais são as metades das larguras e alturas
43   combinedHalfWidths = r1.halfWidth + r2.halfWidth;
44   combinedHalfHeights = r1.halfHeight + r2.halfHeight;
45
46   // Checa por uma colisão no eixo x
```

Continuação: Listagem 8. Funções utilitárias para detecção de colisão.

```
47 if (Math.abs(vx) < combinedHalfWidths) {  
48     // Uma colisão deve estar ocorrendo. Checa por uma colisão no eixo y  
49     if (Math.abs(vy) < combinedHalfHeights) {  
50         // Há de fato uma colisão acontecendo  
51         hit = true;  
52     } else {  
53         // Não há colisão no eixo y  
54         hit = false;  
55     }  
56 } else {  
57     // Não há colisão no eixo x  
58     hit = false;  
59 }  
60 // `hit` será ou `true` ou `false`  
61 return hit;  
62};  
63  
64 function randomInt(min, max) {  
65     return Math.floor(Math.random() * (max - min + 1)) + min;  
66}  
67  
68 function keyboard(keyCode) {  
69     var key = {};  
70     key.code = keyCode;  
71     key.isDown = false;  
72     key.isUp = true;  
73     key.press = undefined;  
74     key.release = undefined;  
75     key.downHandler = function(event) {  
76         if (event.keyCode === key.code) {  
77             if (key.isUp && key.press) key.press();  
78             key.isDown = true;  
79             key.isUp = false;  
80         }  
81         event.preventDefault();  
82     };  
83     key.upHandler = function(event) {  
84         if (event.keyCode === key.code) {  
85             if (key.isDown && key.release) key.release();  
86             key.isDown = false;  
87             key.isUp = true;  
88         }  
89         event.preventDefault();  
90     };  
91     window.addEventListener(  
92         "keydown", key.downHandler.bind(key), false  
93 );  
94     window.addEventListener(  
95         "keyup", key.upHandler.bind(key), false  
96 );  
97     return key;  
98 }
```

- A função `contain()` se encarrega de checar se o objeto está inserido nos limites do container retornando a string referente ao limite que foi atingido, a qual, por sua vez, será usada na função de impressão do objeto de garoto.
- A função `hitTestRectangle()` foi copiada do arquivo do `Bump.js` para evitar a importação de código desnecessário e se encarrega de verificar se os objetos enviados por parâmetro estão em colisão um com o outro. A mesma se encontra autocomentada.
- A função `randomInt()` é bem simples: apenas gera um número aleatório através da função `floor()` da classe `Math` do JavaScript, que retorna o valor inteiro arredondado para baixo do número de ponto flutuante passado por parâmetro (esse, por sua vez, é constituído da multiplicação entre um valor aleatório gerado pela função `random()`, também da classe `Math`, com o resultado da subtração entre o valor máximo e mínimo enviados como argumentos). Ela servirá como uma função utilitária sempre que precisarmos gerar valores aleatórios entre um número mínimo e máximo, como no momento de posicionar os personagens inimigos aleatoriamente no cenário.
- A função `keyboard()` já nos é conhecida e foi apenas reaproveitada na estrutura do documento.

Para essa implementação, o leitor pode considerar a criação de arquivos JavaScript independentes e a respectiva importação do código (a começar por um arquivo de código utilitário que pode ser reaproveitado por várias outras páginas). As **Figuras 2 a 5** ilustram o funcionamento final do jogo após execução no browser. Lembre-se de iniciar a pasta via http-server para que os arquivos de scripts do Pixi sejam importados corretamente.

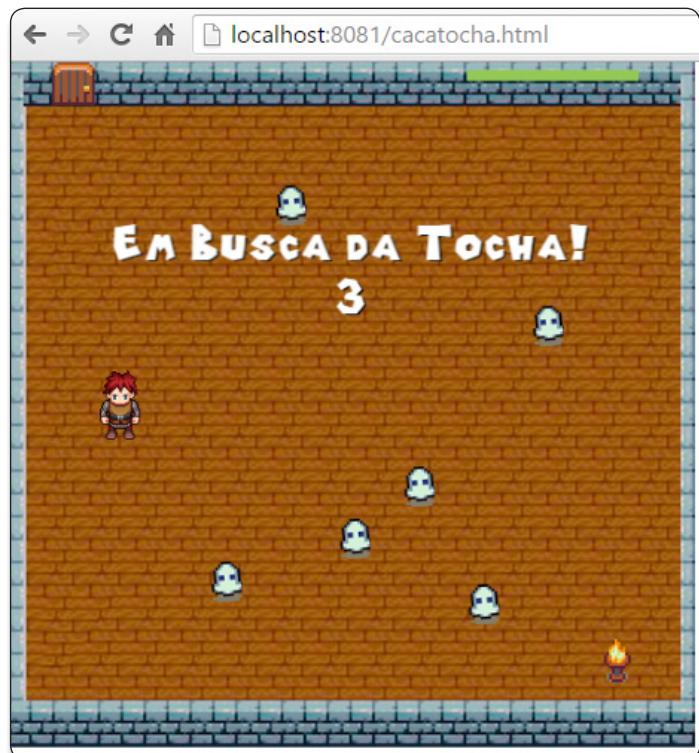


Figura 2. Tela inicial do jogo com contador regressivo

Randomizando os inimigos

O leitor pode opcionalmente randomizar os personagens inimigos ou criar alguma regra inteligente para que os mesmos sejam atualizados ao longo das fases, de acordo com as dificuldades,

tamanhos, etc. O personagem principal também pode ter seu mecanismo de autoescolha de igual forma. Antes de partir para a codificação, selecione algumas imagens de monstros de sua preferência (ou baixe novamente as que temos no arquivo de download do artigo), importe-as no Texture Packer e gere de novo os arquivos PNG e JSON.

A nível de código, nossas alterações acontecerão diretamente na função `setup()`, especificamente no loop que itera sobre a lista de fantasmas. A **Listagem 9** ilustra as mudanças que precisamos realizar para o jogo se assemelhar à **Figura 6**. Note que apenas mapeamos cada um dos inimigos que estão no arquivo JSON gerado dentro do vetor `personagensInimigos`, recuperando apenas um dos valores a cada iteração do loop por meio da função utilitária

que vimos, `randomInt()`, que, por sua vez, receberá um valor mínimo de 0 e máximo do tamanho do vetor (decrecido de 1 já que vetores começam de zero). No fim (linha 10), buscamos do objeto `id` o valor escolhido para exibir no objeto fantasma.

Mudando textures de sentido

Para tornar o cenário mais realista, além de randomizar os personagens inimigos e exibi-los em posições aleatórias na tela, seria interessante que tivéssemos uma textura diferente para cada sentido no qual os mesmos navegam, isto é, algo que mostre ao jogador que o personagem está de frente ou de costas. Para fazer isso, precisaremos trabalhar sobre a propriedade `texture` dos objetos de sprites, modificando seu valor de acordo com os valores do arquivo JSON.

Listagem 9. Código novo que itera sobre a lista de fantasmas, randomizando os personagens.

```
01 // Um vetor para armazenar todos os fantasmas
02 fantasmas = [];
03
04 var personagensInimigos = ["fantasma.png", "aranha.png", "morcego.png",
  "gosma.png"];
05
06 // Cria quantos fantasmas a variável `numDeFantasmas` informar
07 for (var i = 0; i < numDeFantasmas; i++) {
08   // Cria um fantasma
09   var persoAleatorio = personagensInimigos[randomInt(
  0, personagensInimigos.length - 1)];
10   var fantasma = new Sprite(id[persoAleatorio]);
11
12 // ...
```



Figura 3. Personagem sendo atingido e barra de saúde decrescida

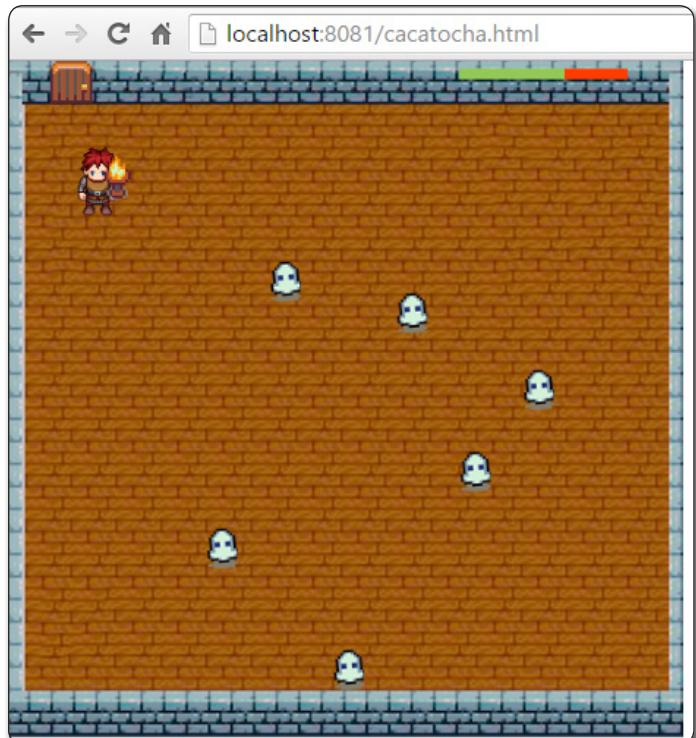


Figura 4. Personagem com a tocha em busca da saída

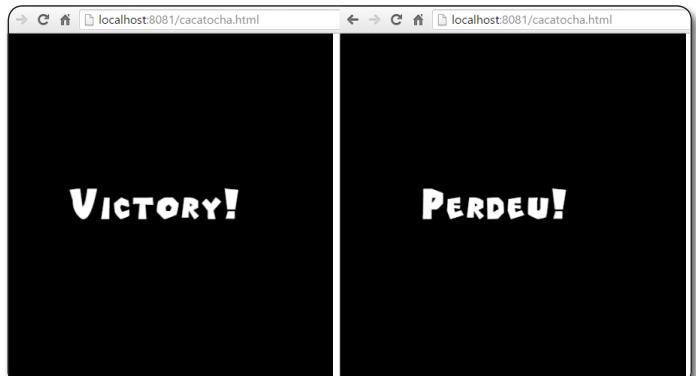


Figura 5. Telas de vitória e derrota, respectivamente



Figura 6. Personagens inimigos randomizados

Você já deve ter notado que tanto no arquivo PNG quanto no JSON que importou do código fonte final temos imagens dos referidos personagens de costas. Isso será útil para atingir nosso objetivo aqui. Portanto, atualize novamente seus arquivos para que começemos as mudanças criando um novo vetor de *personagensInimigosVolta* que, por sua vez, guardará os mesmos objetos de inimigos nas mesmas posições, porém com o sufixo *-volta*, que usaremos para nos localizar sobre qual das imagens exibir:

```
var personagensInimigosVolta = ["fantasma-volta.png", "aranha-volta.png", "morcego-volta.png", "gosma-volta.png"];
```

Note que tanto esse vetor como o de inimigos de frente devem ser movidos para fora da função *setup()*, assim poderemos usá-los em outras ocasiões ao longo do código. Em seguida, no mesmo loop que iteramos os objetos de fantasmas, vamos modificar suas linhas iniciais para o código demonstrado na **Listagem 10**. Observe que, como agora temos dois vetores para manusear e precisamos garantir que o objeto apareça com a imagem certa de acordo com a direção que o mesmo assumir quando for criado, precisamos checar qual o valor da variável *direction* (1 para baixo e -1 para cima) definindo, assim, na variável *getPersonagem* o vetor que deverá ser usado para recuperar cada objeto randômico de fato. O código de recuperação e criação do Sprite (linhas 11 e 12) continuam o mesmo, exceto pela referência agora à referida variável.

Na linha 13 configuramos uma nova propriedade dinâmica *id* em cada objeto de fantasma, tendo como valor a posição do

personagem selecionado no vetor. Isso nos será útil mais adiante quando precisarmos mudar a imagem quando o personagem atingir uma das extremidades.

Nota

No JavaScript é possível criar variáveis em objetos de forma dinâmica, à medida que as mesmas forem necessárias. Esse comportamento difere da maioria das linguagens compiladas, que definem tal estrutura antes de forma fixa e necessitam ter seu código recompilado caso haja alguma mudança desse tipo.

Listagem 10.

Novo código de criação randômica dos personagens inimigos.

```
01 // Cria quantos fantasmas a variável 'numDeFantasmas' informar
02 for (var i = 0; i < numDeFantasmas; i++) {
03     // Cria um fantasma
04     var getPersonagem;
05     if (direction === 1) {
06         getPersonagem = personagensInimigos;
07     } else {
08         getPersonagem = personagensInimigosVolta;
09     }
10
11     var persoAleatorio = getPersonagem[randomInt
12     (0, getPersonagem.length - 1)];
13     fantasma = new Sprite(id[persoAleatorio]);
14     fantasma.id = getPersonagem.indexOf(persoAleatorio);
15
16     var x = spacing * i + xOffset;
17     // ...
18
19}
```

Na **Listagem 11**, você encontra o restante do código para atualizar as imagens das texturas dos personagens inimigos quando os mesmos atingirem umas das extremidades do topo ou da base do cenário. O mesmo deve ser inserido no *forEach()* dos fantasmas da função *play()* logo após o teste de mudança de velocidade.

A verificação é bem simples: caso o fantasma tenha colidido com o topo do cenário, mudamos a textura do inimigo para frente, caso contrário mudamos para costas. É isso, nossa implementação finalizou. Basta agora recarregar o jogo no browser e você verá algo semelhante à **Figura 7**.

O mesmo pode ser feito para o personagem do garoto, porém, neste caso, como o mesmo precisará se mover para as quatro direções, precisamos mapear imagens para todas elas. Para simplificar, a imagem que você importou do código fonte do projeto disponibiliza apenas as direções baixo, direita e esquerda. Atualize-as no Texture Packer e substitua os arquivos JSON e PNG nos respectivos diretórios.

Após isso, nossa lógica consistirá em apenas adicionar as mudanças de texturas nos eventos de pressionamento das teclas de setas do teclado na função *configKeys()*, tal como demonstra a **Listagem 12**. Veja que as respectivas funções de *release* das keys precisam de igual forma estar mapeadas para que o garoto retorne à posição natural quando as teclas forem soltas.

Desenvolvimento de jogos web com Pixi.js – Parte 3

Mas além disso, também precisaremos verificar quando o personagem atinge o objeto de tocha, levando em consideração que as suas dimensões são menores quando ele está de lado, portanto nosso código precisa lidar com esse reposicionamento dos refe-



Figura 7. Personagens exibidos de frente e de costas



Figura 8. Personagem se movendo ao longo do cenário

ridos objetos. Para isso, no início do script onde declaramos as variáveis globais do código, adicione uma nova tal como:

```
getTacha = false;
```

Em seguida, na função play(), devemos mudar o teste da colisão para verificar antes se a variável `getTacha` está com valor `true`, assim evitamos a detecção de colisão entre o garoto e a tocha já que para este caso em específico isso não poderá ocorrer. A **Listagem 13** ilustra como esse teste ficará agora. O resultado pode ser conferido na **Figura 8**.

Listagem 11. Código para atualizar as texturas quando personagem atingir topo ou base.

```
01 // Se o fantasma atinge o top ou bottom do stage, reverte sua direção
02 if (fantasmaColideParede === "top" || fantasmaColideParede === "bottom") {
03     fantasma.vy *= -1;
04 }
05
06 if (fantasmaColideParede === "top") {
07     fantasma.texture = id[personagensInimigos[fantasma.id]];
08 } else if (fantasmaColideParede === "bottom") {
09     fantasma.texture = id[personagensInimigosVolta[fantasma.id]];
10 }
11
12 // Testa se houve uma colisão. Se qualquer um dos inimigos estiver tocando o
13 // garoto, configura o 'garotoColide' para 'true'
14 if (hitTestRectangle(garoto, fantasma)) {
15     garotoColide = true;
16 }
```

Listagem 12. Código para mapear mudança de textura do garoto quando em movimento.

```
01 left.press = function() {
02     // Muda a velocidade do garoto quando a tecla é pressionada
03     garoto.vx = -5;
04     garoto.vy = 0;
05     garoto.texture = id["garoto-left.png"];
06 };
07 left.release = function() {
08     if (!right.isDown && garoto.vy === 0) {
09         garoto.vx = 0;
10         garoto.texture = id["garoto.png"];
11     }
12 };
13
14 right.press = function() {
15     garoto.vx = 5;
16     garoto.vy = 0;
17     garoto.texture = id["garoto-right.png"];
18 };
19 right.release = function() {
20     if (!left.isDown && garoto.vy === 0) {
21         garoto.vx = 0;
22         garoto.texture = id["garoto.png"];
23     }
24 };
```

Listagem 13. Teste para verificar se o garoto já pegou a tocha.

```
01 // Checa se há uma colisão entre o garoto e a tocha
02 if (getTacha || hitTestRectangle(garoto, tocha)) {
03     // Se a tocha estiver estiver tocando o garoto, a centraliza sobre o garoto
04     tocha.x = garoto.x + 20;
05     tocha.y = garoto.y + 6;
06     getTacha = true;
07 }
```

Esse tipo de configuração constitui apenas uma pequena porcentagem diante das possibilidades de animação para com objetos e personagens. O Pixi fornece uma série de mecanismos para automatizar esse processo sem que precisemos estar o tempo inteiro testando a posição dos mesmos no cenário ou detectando o pressionamento das teclas de movimento.

Além disso, também vimos que o JavaScript é um ótimo aliado na hora de implementar quaisquer tipos de códigos assíncronos, temporizadores ou contadores regressivos para que seu jogo

alcance cada vez mais um tom profissional. Por se tratar de uma aplicação web comum, o leitor pode ficar à vontade inclusive para adicionar quaisquer outras bibliotecas JavaScript de seu agrado que possam lhe ajudar a aprimorar o jogo como um todo. Todavia, tome sempre bastante cuidado com o excesso de scripts carregados na página, pois, dependendo da finalidade do seu jogo, isso poderá atrapalhar no carregamento e SEO da página, portanto avalie bem sempre as opções e pese se realmente tais arquivos são necessários. Por fim, não esqueça de checar também se o próprio Pixi.js já não disponibiliza nenhum dos recursos por padrão, para evitar duplicidade de código desnecessariamente; nessas horas uma boa lida na documentação do framework ajuda muito. Bons estudos!

Autor



Júlio Sampaio



É 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:

Biblioteca de colisões 2D para o Pixi.

<https://github.com/kittycatattack/bump>

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.

Suporte 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 | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486