

Google CHARTS

**GRÁFICOS
DINÂMICOS EM
JAVASCRIPT
E HTML5**



Google Dart

Aprenda a nova linguagem
de scripts do Google

Meteor.js

Crie aplicações web integrando
Node.js e MongoDB

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)

Consultor Técnico

Daniella Costa (daniella.devmedia@gmail.com)

Produção

Jornalista Responsável Kaline Dolabella - JP24185

Capa e Diagramação Romulo Araujo

Atendimento ao leitor

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

Publicidade

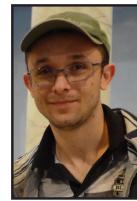
publicidade@devmedia.com.br – 21 3382-5038

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 a 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 Boas Práticas

04 – Google Charts: Criando Gráficos dinâmicos com JavaScript

[Júlio Sampaio]

Artigo no estilo Solução Completa

17 – Meteor.js: Construindo aplicações web com Node.js e MongoDB

[Jorge Rodrigues]

Conteúdo sobre Boas Práticas, Conteúdo sobre Novidades

29 – Primeiros passos com a Google Dart

[Julio Sampaio]

Conteúdo sobre Boas Práticas

41 – Testes automatizados com o Framework Selenium

[Sueila Sousa]

Google Charts: Criando Gráficos dinâmicos com JavaScript

Veja como incorporar todo o potencial da API de gráficos do Google nas suas aplicações web

Por muito tempo a utilização de gráficos dinâmicos, muitas vezes gerados a partir de extensas fontes de dados com inúmeros cruzamentos de nível relacional nas bases de dados das aplicações, era uma tarefa reclusa somente aos ambientes de desenvolvimento server side ou desktop, que, por sua vez, faziam uso de linguagens de programação fortemente tipadas como o Java ou C#. Além disso, eles não podiam existir sozinhos, mas sim pertencer a alguma estrutura de relatório, com geração sempre feita em arquivos PDFs. Para se atingir tal objetivo, principalmente em aplicações web cross platform que têm uma camada de apresentação baseada em um browser, os passos são sempre os mesmos:

1. Primeiro desenvolve-se um layout, um template, através de uma ferramenta de desenho adaptada para aquela tecnologia de relatórios e que gere um código XML no fim, como o iReport para relatórios em Java, e o Crystal Reports para relatórios em .NET, por exemplo.

2. Uma vez criado o esqueleto com os devidos campos em branco e variáveis de referência apontando para cada um, que servem como lacunas a serem preenchidas pela linguagem dinamicamente, é hora de gerar toda a lógica e reunir os dados para enviar ao mesmo layout, via código ou acesso direto a um banco de dados.

3. Finalmente, se a aplicação necessitar imprimir o relatório em uma tela no navegador, serializamos todos os bytes do relatório e os enviamos em vários pacotes (processo esse que é abstraído pela tecnologia server side utilizada) via HTTP. O browser, ao receber tais pacotes, irá processar o MIME type (vide **BOX 1**) do arquivo final e executar a ação padrão para o mesmo (fazer o download do arquivo, abri-lo numa ferramenta desktop padrão, abri-lo num plugin do próprio browser, etc.).

Fique por dentro

Desenvolver gráficos é uma tarefa obrigatória em inúmeras aplicações corporativas e a maioria das empresas já não adotam mais soluções baseadas no back-end. Frameworks JavaScript como o Google Charts oferecem poderosos recursos para construir gráficos de todo tipo e de forma rápida, produtiva e performática, tudo usando apenas JavaScript, CSS e HTML5. Além de ser multiplataforma, o Google Charts ainda se integra facilmente com Web Services e serviços de frameworks como AngularJS, jQuery, Polymer, dentre outros.

Neste artigo exploraremos os mais diversos recursos dessa biblioteca, expondo situações do cotidiano, além de entender qual gráfico atende melhor a cada situação. Mostraremos como modificar estilo, proporções, estrutura, bem como adaptar os layouts ao novo conceito de design do Google: O Material Design.

BOX 1. MIME type

MIME type quer dizer Internet Media Type, ou Tipo de Mídia de Internet, e serve para identificar o tipo de dado que um determinado arquivo contém. Os usos mais comuns são para softwares clientes de e-mail, navegadores web e mecanismos de buscas. Geralmente eles são divididos em tipos e subtipos. Por exemplo, usamos os MIME types para designar nomes de arquivos quando os importamos em uma página HTML, como os arquivos de CSS (text/css) e JavaScript (text/javascript). Neste caso, o text é o tipo e css-javascript são os subtipos.

São muitos passos para se atingir um objetivo deveras simples, isso sem falar da sobrecarga gerada na aplicação ao trafegar informações de download (e downloads não são a especialidade do HTTP) sempre que o usuário solicitar uma nova impressão. Alguns desenvolvedores optaram então pelo uso da tecnologia Ajax no client side para aumentar a performance das requisições, uma vez que elas serão feitas de forma assíncrona.

Alguns frameworks como o JSF ou ASP.NET dão suporte a esse tipo de funcionalidade, porém, isso só mascara o problema, pois os bytes continuarão sendo trafegados.

A solução encontrada pela maioria das empresas para diminuir esse gargalo do server side foi focar no que há de mais crucial para as aplicações: os dados, e deixar todo o resto com o JavaScript (que também é muito leve). Representar dados em aplicações front-end é muito simples e leve, basta usar objetos JSON ou XML para fazer a transição e o efeito é altamente performático.

Inúmeros são os plug-ins e bibliotecas JavaScript que fornecem recursos como esse, dentre os quais destacam-se uma gama de plug-ins do jQuery (como o HighCharts), o Flotr2 e o Google Charts.

O maior desafio de uma empresa que lança uma ferramenta desse tipo é a qualidade (visual e do código) e a quantidade de gráficos disponíveis, uma vez que os usuários se tornam cada vez mais exigentes e as informações crescem exponencialmente em número e complexidade dependendo da área de negócio trabalhada. O Google Charts atende muito bem a essas exigências, principalmente por englobar em sua criação a realidade de uma das empresas que mais usam gráficos: o próprio Google.

A ferramenta do Google Charts provê uma série de recursos para construir gráficos de forma fácil e adicioná-los às suas páginas web de forma responsiva, adaptando o design à realidade das mesmas. Existem basicamente dois tipos de gráficos que podem ser trabalhados: os **estáticos** e os **interativos**. A principal diferença está na forma como você deseja que seu gráfico interaja com o usuário, pois os gráficos estáticos fornecerão apenas uma imagem dos dados dinâmicos recebidos, ao passo que os interativos disponibilizarão eventos JavaScript (como click, focus, mouseover/out/in, etc.) para efetuar ações via código quando os mesmos forem disparados.

Neste artigo trataremos de explorar os recursos mais importantes do Google Charts, desenvolvendo ambos os tipos de relatório citados em diversos formatos (gráficos de linha, barras, pizza, geográficos, etc.), bem como a adaptação dos mesmos ao novo conceito de design do Google: o Material Design.

Primeiros gráficos

O Google Chart fornece duas formas de geração de gráficos: via URL e via código JavaScript. O primeiro tipo fornece uma URL universal que, via método GET HTTP, retorna uma imagem estática do gráfico gerada em detrimento dos parâmetros passados pela mesma URL, que constituem a própria fonte de dados. Essa opção, por sua vez, permite somente a geração de gráficos simples, que não envolvem tomadas de decisão ou estruturas condicionais. Já a segunda opção se dá através da importação das bibliotecas da API, sobrescrita das funções núcleo e fornecimento dos dados via *mocks* ou serviços remotos.

Vejamos a URL de exemplo representada a seguir, cujo gráfico gerado é ilustrado pela **Figura 1**:

```
http://chart.apis.google.com/chart?cht=p3&chs=550x300&chd=t:5,4,3,6&chl=PCs|Geladeiras|TVs|Outros&cht=%C3%9AltimasVendas&chco=8cc53e
```

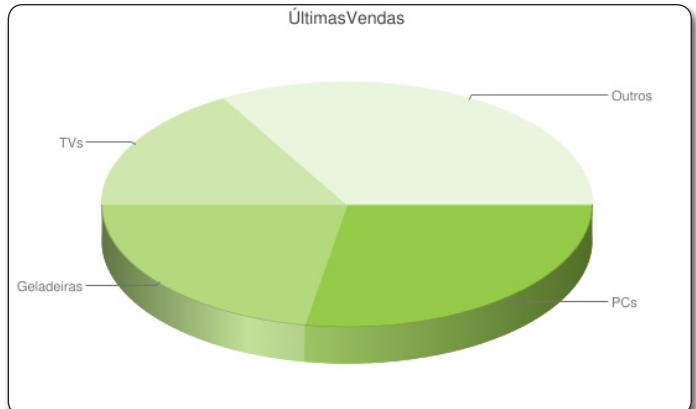


Figura 1. Gráfico de exemplo gerado via URL

Vejamos alguns detalhes:

- Esse recurso está hospedado por padrão no site oficial do Google Charts (vide seção **Links**), portanto a primeira parte da URL referente ao domínio representa o mesmo site. Por ser um serviço que executa em GET no HTTP, a performance em si é aumentada;
- O primeiro parâmetro (**cht**, de *chart type*), definido após o sinal de interrogação, é referente ao tipo do gráfico (neste caso, pizza em 3D = p3). Você pode encontrar uma lista dos tipos disponíveis também no site oficial;
- O parâmetro **chs** (de *chart size*) define as dimensões (largura x altura) do gráfico, sempre em pixels;
- O parâmetro **chd** (de *chart data*) define os valores para cada “pedaço” do gráfico. Esses valores devem sempre ser particionados em forma percentual, uma vez que a API somará tudo e definirá o percentual de cada parte no todo;
- O parâmetro **chl** (de *chart labels*) se encarrega de receber o vetor com as legendas que serão exibidas para cada item do gráfico. É importante que os valores estejam em mesmo número e ordem que os definidos no parâmetro chd, senão você receberá um erro na impressão do gráfico;
- O parâmetro **cht** (de *chart title*) recebe o valor do título do gráfico;
- Por fim, o parâmetro **chco** (de *chart color*) recebe a cor em hexadecimal (sem o sinal de #) que será exibida no gráfico. O valor “8cc53e” é referente à cor padrão da logo da DevMedia. Note que a mesma cor é distribuída em tons mais claros e escuros para diferenciar as partes do gráfico. Esse tipo de comportamento é padrão no Google Charts, mas se você desejar atrelar uma cor diferente a cada pedaço, basta configurar o parâmetro com os valores em forma de vetor, por exemplo: `chco=8cc53e|ff0000|1112233|123456`.

Nota

Todos os gráficos gerados pelo Google Charts são renderizados usando SVG (Scalable Vector Graphics) ou VML (Vector Markup Language), que são linguagens de marcação baseadas em vetores usadas para representar gráficos bidimensionais em páginas web. O Google vem trabalhando para incluir também o Canvas da HTML5.

Essa opção, apesar das limitações, é bem flexível e simples, o que permite que qualquer usuário, inclusive os que não têm conhecimentos de programação, possa usá-la. E esse foi o objetivo do Google ao disponibilizá-la. Se você desejar dinamizar o conteúdo, basta efetuar chamadas seguidas à mesma URL modificando apenas os parâmetros, lembrando que essa URL precisa estar sempre dentro do atributo src da tag img na sua página HTML, o que te permite inclusive modificar seu conteúdo via JavaScript, jQuery, etc.

O Google também oferece uma página onde você pode desenhar os gráficos online enquanto ela mesma gera a URL com as respectivas opções, trata-se do **Google Live Chart Playground** (vide seção **Links**).

Já em relação ao uso da segunda opção, via código JavaScript, antes de iniciar as implementações de fato, precisamos configurar o ambiente, bem como a estrutura de diretórios que usaremos para padronizar o acesso aos caminhos relativos de dentro das páginas HTML. Para isso, crie uma estrutura semelhante à ilustrada pela **Listagem 1** na sua máquina.

Listagem 1. Estrutura de diretórios padrão do projeto.

```
google-charts-devmedia
|---css
|---index.css
|---js
|---index.js
|---img
|---index.html
```

Note que os três arquivos HTML, CSS e JS devem ser criados também para conter o código que vamos desenvolver inicialmente.

Feito isso, existem três passos básicos para se configurar qualquer gráfico com o Google Charts, a saber:

1. A API Google JSAPI: Essa API comporta todas as funções Ajax e JavaScript necessárias para carregar os gráficos, bem como a lógica para montar, distribuir e dinamizar os dados. Para incluí-la na sua aplicação basta inserir o trecho de importação de script a seguir na sua página HTML:

```
<!-- Carrega a AJAX API -->
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
```

2. A biblioteca Google Visualization: Essa biblioteca define as classes e funções de núcleo e utilitárias da API. Temos, por exemplo, as classes 'DataTable' para lidar com os dados em forma de tabela, e 'Query' para busca de dados nos provedores, além de 'error handlers' para ajudar a identificar e exibir erros nas páginas. A forma de importação da mesma, entretanto, se dá de forma diferente das demais: em vez de importar um arquivo diretamente dos servidores do Google, precisamos carregar um módulo (**visualization**, para ser mais específico) via JavaScript, informando a versão e a lista de pacotes que queremos importar:

```
<script type="text/javascript">
  google.load('visualization','1.0',{ 'packages':[ <lista_dos_nomes_pacotes> ]});</script>
```

3. O código para o gráfico em si: Também é preciso definir o código de criação do gráfico propriamente dito, através do uso de classes como PieChart, BarChart, etc. Para isso é preciso sobrecrever o método **setOnLoadCallback()** com a função que deverá ser chamada para construir o gráfico em si. Veja o código que você precisa usar para isso:

```
<script type="text/javascript">
  google.setOnLoadCallback(desenharGrafico );
</script>
```

Nota

O leitor precisará ter obrigatoriamente uma conexão com a internet para testar os exemplos, visto que os arquivos JavaScript, de CSS e imagens serão carregados diretamente dos servidores Google.

No passo 2 vimos que o Google Charts se utiliza do arquivo JavaScript importado no passo 1 para carregar os módulos que precisamos para os gráficos. Esse tipo de recurso modularizado já é comum em inúmeros frameworks JavaScript, como o Grunt, CoffeeScript, etc. pois aumenta a performance da aplicação ao carregar somente o que é necessário naquele momento. Em outras palavras, imagine que você tem um arquivo JS de 2 MB de tamanho final, que contempla todo o código para criar qualquer sorte de gráfico:

- A vantagem é que esse arquivo será carregado apenas uma vez, logo não precisaremos efetuar mais chamadas ao servidor quando precisarmos acessar um módulo X, Y ou Z;
- A desvantagem é que ele é muito pesado e isso gera sobrecarga, deixando a aplicação mais lenta e menos visível às ferramentas de busca.

Para solucionar isso, a API divide o arquivo em vários menores e, quando um determinado gráfico precisa ser usado (pizza, linhas, colunas, etc.), basta importar o módulo correspondente ao mesmo. No nosso exemplo, cada módulo teria o equivalente a 15 KB, caracterizando assim um ganho enorme para a aplicação.

Vejamos então a implementação necessária para criar um gráfico de colunas via JavaScript. Na **Listagem 2** temos o código de um gráfico onde cruzamos a média salarial entre várias profissões de TI, que será refletido na **Figura 2**.

A figura está dividida em duas partes: a primeira representa o gráfico normal, e a segunda o gráfico com a coluna destacada ao pôr o mouse sobre a mesma. Vejamos alguns detalhes da implementação:

- Nas linhas 8, 11 e 14 vemos a implementação obrigatória dos três passos vistos, respectivamente.
- Na linha 17 temos a função **criarGrafico()** que conterá o código para criação do objeto de gráfico (neste caso um objeto DataTable, linha 19), bem como criação das colunas e dados.
- Nas linhas 22 e 23 temos a criação das colunas com dois parâmetros: o tipo de dado (string, number, char, boolean, etc.) e a descrição das mesmas. Esta última só será usada se você explicitar alguma ação interativa nos eventos JavaScript.

Listagem 2. Código para criação de gráfico de colunas.

```
01 <!DOCTYPE>
02 <html>
03 <head>
04   <title>Google Charts - DevMedia</title>
05   <meta charset="utf-8">
06
07   <!-- Carrega a AJAX API -->
08   <script type="text/javascript" src="https://www.google.com/jsapi"> </script>
09   <script type="text/javascript">
10     // Carrega a API da biblioteca Visualization e add os nomes dos pacotes.
11     google.load('visualization','1', {packages: ['columnchart']});
12
13   // seta a função de callback
14   google.setOnLoadCallback (criarGrafico);
15
16   // função de callback
17   function criarGrafico() {
18     // cria o objeto DataTable que falamos antes
19     var dataTable = new google.visualization.DataTable();
20
21     // Define as colunas
22     dataTable.addColumn('string','Profissionais TI 2015');
23     dataTable.addColumn('number','Salários');
24
25     // Define os dados a serem exibidos
26     dataTable.addRows([['Designer', 1500], ['Programador', 2500],
27                       ['Arquiteto de Software', 4750], ['Gerente de Projetos', 7500]]);
28
29     // Instancia o nosso objeto chart
30     var chart = new google.visualization.ColumnChart
31       (document.getElementById('chart'));
32
33     // Define as opções para visualização
34     var options = {width: 600, height: 350, is3D: true,
35                   title: 'Média de Salários em TI'};
36
37     // desenha o nosso gráfico de fato
38     chart.draw(dataTable, options);
39
40   </script>
41 </head>
42 <!-- Div para o nosso grafico -->
43 <div id="chart"></div>
44 </body>
45 </html>
```

- Na linha 26 é onde criamos os dados fixos para teste do gráfico. O método `addRows()` recebe sempre um vetor com os valores na ordem e quantidade das colunas previamente definidas.

- Na linha 29 criamos um novo objeto do tipo `ColumnChart` que representa o objeto de gráfico em si, passando como parâmetro a div HTML que criamos na linha 43. É dentro dela que o Google Charts irá enxertar todo o HTML do gráfico.

- Na linha 32 definimos o objeto `options`. Este objeto é opcional e serve para configurar as propriedades de estrutura e estilo do gráfico, como dimensões, título, etc.

- Por fim, na linha 35, chamamos o método `draw()` que se encarregará de fazer todo o trabalho de desenho, passando o objeto de dados `dataTable` e o objeto de `options`.

Note que na linha 11, onde definimos o tipo do gráfico que queremos desenhar, você pode substituir o valor da propriedade `packages` por outro tipo, como o gráfico em barras. Veja como ficaria a implementação:

```
google.load('visualization','1', {packages: ['corechart']});
```

Agora é só salvar o arquivo e reexecutar a página no browser. O resultado deverá ser igual ao da **Figura 3**.



Figura 2. Gráfico de colunas de exemplo gerado via JavaScript

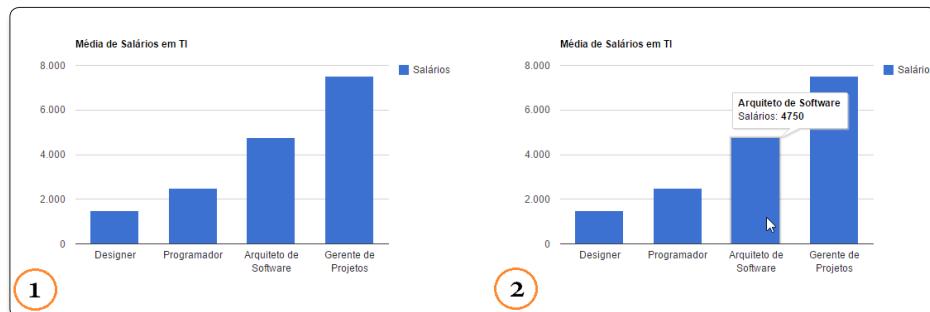


Figura 3. Gráfico de barras de exemplo gerado via JavaScript

Perceba como uma pequena alteração no código traz um resultado enorme para o visual do gráfico. O valor “corechart” é usado para associar o estilo ao novo padrão do Google: o Material Design (vamos explorá-lo mais à frente). Além

disso, agora temos a exibição da descrição em um balão suspenso para cada barra do gráfico com o respectivo valor. O leitor pode testar outros valores e ver como a API se comporta, mas lembre-se que os gráficos precisam ter a mesma

Google Charts: Criando Gráficos dinâmicos com JavaScript

estrutura de cruzamento dos dados, neste caso dados representados pelos eixos x e y do plano cartesiano.

Além de textos e números, também é possível criar gráficos múltiplos, manipular outros tipos de dados como datas e valores percentuais, definir posições diferentes para legendas e subtítulos, dentre outros. Suponha que, no mesmo exemplo, você deseja criar uma coluna ou barra dupla para representar as médias mínima e máxima de salários. Esses gráficos são chamados de *Dual-Y charts*, justamente pela dualidade de colunas no eixo Y. Crie então uma nova página HTML e adicione o conteúdo da **Listagem 3** à mesma.

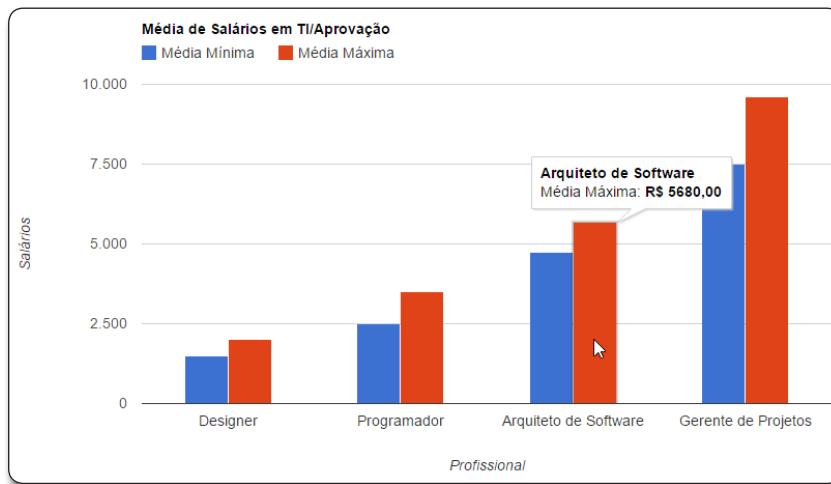


Figura 4. Gráfico de barras duplas gerado via JavaScript

O resultado pode ser verificado na **Figura 4**.

Perceba que a estrutura base do gráfico é a mesma, porém fizemos várias alterações, a saber:

- Na linha 22 adicionamos uma nova coluna ao objeto gráfico. Sempre que for necessário fazer isso, a ordem de tipo do dado e descrição deve ser respeitada.
- A partir da linha 24 estamos populando o nosso vetor de linhas com as mesmas labels de profissionais, porém os valores agora estão com uma notação diferente. A variáveis **v** e **f** descrevem o valor numérico e a descrição que deverá aparecer no gráfico, respectivamente. Esse recurso funciona de forma semelhante aos alias de bancos de dados, onde em vez de exibirmos o nome da própria coluna (que geralmente é mais técnico que apresentável) criamos um valor mascarado que será exibido no lugar. Através dessa máscara, você consegue incutir qualquer texto pré-formatado e o resultado fica bem mais elegante, conforme demonstrado no balão descriptivo do gráfico final.
- A partir da linha 31 definimos o objeto **options**, dessa vez com algumas configurações novas:
 - A propriedade **position** definida no atributo **legend** da linha 34 modifica a posição das caixas de legendas no gráfico. Os valores possíveis são *top*, *bottom*, *left* ou *right*. Já **maxLines**, na mesma linha, define o número máximo de linhas que a legenda deverá ocupar na impressão.

Listagem 3. Criando gráfico duplo.

```
01 <!DOCTYPE>
02 <html>
03   <head>
04     <title>Google Charts - DevMedia</title>
05     <meta charset="utf-8">
06
07     <!-- Carrega a AJAX API -->
08     <script type="text/javascript" src="https://www.google.com/jsapi"> </script>
09     <script type="text/javascript">
10       // Carrega a API da biblioteca Visualization e add os nomes dos pacotes.
11       google.load('visualization', '1', {'packages': ['corechart']});
12
13       // seta a função de callback
14       google.setOnLoadCallback(criarGrafico);
15
16       // função de callback
17       function criarGrafico() {
18         var dataTable = new google.visualization.DataTable();
19
20         dataTable.addColumn('string', 'Profissionais TI 2015');
21         dataTable.addColumn('number', 'Média Mínima');
22         dataTable.addColumn('number', 'Média Máxima');
23
24         dataTable.addRows([
25           ['Designer', {v: 1500, f:'R$ 1500,00'}, {v: 2000, f:'R$ 2000,00'}],
26           ['Programador', {v: 2500, f:'R$ 2500,00'}, {v: 3500, f:'R$ 3500,00'}],
27           ['Arquiteto de Software', {v: 4750, f:'R$ 4750,00'},
28             {v: 5680, f:'R$ 5680,00'}],
29           ['Gerente de Projetos', {v: 7500, f:'R$ 7500,00'}, {v: 9600, f:'R$ 9600,00'}],
30         ]);
31         var options = {
32           width: 1000,
33           height: 500,
34           legend: {position: 'top', maxLines: 3},
35           title: 'Média de Salários em TI/Aprovação',
36           hAxis: {
37             title: 'Profissional'
38           },
39           vAxis: {
40             title: 'Salários'
41           },
42           bar: { groupWidth: "50%" }
43         };
44
45         var chart = new google.visualization.ColumnChart(
46           document.getElementById('chart'));
47
48         chart.draw(dataTable, options);
49       }
50     </script>
51
52   </head>
53
54   <body>
55     <!-- Div para o nosso grafico -->
56     <div id="chart"></div>
57   </body>
58 </html>
```

- Os atributos **hAxis** e **vAxis**, servem para configurar propriedades específicas dos valores exibidos nos eixos horizontal e vertical, respectivamente, tais como formatação, títulos, valores máximos e mínimos, etc. Neles, definimos os títulos dos dois eixos: "Salários" e "Profissional", conforme observado no gráfico final.
- O atributo **bar** da linha 42 é exclusivo para gráficos em barras e serve para definir o tamanho máximo das barras, através da propriedade **groupWidth**. Note como as barras estão 50% menores em relação ao exemplo anterior.

Assimilado esse conhecimento, é muito fácil criar estruturas novas. Por exemplo, agora você precisa gerar um gráfico com o número de atividades entregues e pendentes em um determinado projeto de TI, fazendo um rastreio ao longo do dia útil de trabalho. Vejamos a implementação da **Listagem 4** (apenas com o código JavaScript, a HTML permanece igual).

O resultado dessa implementação pode ser observado na **Figura 5**. Vejamos algumas observações:

- Na linha 11 temos a adição de um novo tipo de dado: **timeofday**, que funciona como o Date do JavaScript.
- A maior mudança acontece na linha 15, onde estamos definindo os objetos de data/hora, e número de atividades. Note que aqui também usamos as variáveis **v** e **f** para criar os alias porém,

as datas devem ser definidas sempre em forma de vetor com três valores: hora, minuto e segundo, respectivamente.

- No objeto **options**, configuramos o eixo horizontal (**hAxis**) na linha 41 com dois novos atributos:

- **format**: define a regra de formatação da data, que será sempre hora:minuto.

- **viewWindow**: define os valores mínimo e máximo para a representação das horas no eixo horizontal.

Existe ainda uma segunda forma de se implementar os mesmos gráficos, através da função **arrayToDataTable()** do Google Charts. Através dela podemos reduzir a quantidade de código por meio das **roles** (papéis) que incutem formatação no momento de criação

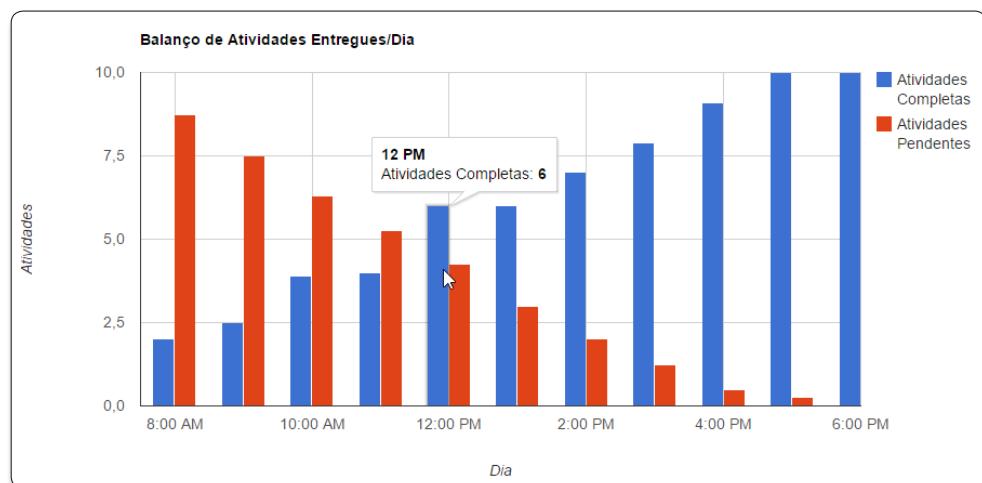


Figura 5. Gráfico de barras usando rastreio de horas

Listagem 4. Exemplo de gráfico em barras usando rastreio de horas.

```

01 <script type="text/javascript">
02 // Carrega a API da biblioteca Visualization e add os nomes dos pacotes.
03 google.load('visualization','1', {'packages': ['corechart']});
04
05 // seta a função de callback
06 google.setOnLoadCallback (criarGrafico);
07
08 // função de callback
09 function criarGrafico() {
10     var dataTable = new google.visualization.DataTable();
11     dataTable.addColumn('timeofday', 'Dia');
12     dataTable.addColumn('number', 'Atividades Completas');
13     dataTable.addColumn('number', 'Atividades Pendentes');
14
15     dataTable.addRows([
16         [{v: [8, 0, 0], f:'8 AM'}, 2, 8.75],
17         [{v: [9, 0, 0], f:'9 AM'}, 2.5, 7.5],
18         [{v: [10, 0, 0], f:'10 AM'}, 3.9, 6.3],
19         [{v: [11, 0, 0], f:'11 AM'}, 4, 5.25],
20         [{v: [12, 0, 0], f:'12 PM'}, 6, 4.25],
21         [{v: [13, 0, 0], f:'1 PM'}, 6, 3],
22         [{v: [14, 0, 0], f:'2 PM'}, 7, 2],
23         [{v: [15, 0, 0], f:'3 PM'}, 7.89, 1.25],
24         [{v: [16, 0, 0], f:'4 PM'}, 9.1, 0.5],
25         [{v: [17, 0, 0], f:'5 PM'}, 10, 0.25],
26         [{v: [18, 0, 0], f:'5 PM'}, 10, 0],
27     ]);
28
29     var options = {
30         width: 1000,
31         height: 500,
32         title: 'Balanco de Atividades Entregues/Dia',
33         hAxis: {
34             title: 'Dia',
35             format: 'h:mm a'
36         },
37         viewWindow: {
38             min: [7, 30, 0],
39             max: [18, 0, 0]
40         },
41         vAxis: {
42             title: 'Atividades'
43         }
44     };
45
46     var chart = new google.visualization.ColumnChart(
47         document.getElementById('chart'));
48
49     chart.draw(dataTable, options);
50 }
51 </script>
```

Google Charts: Criando Gráficos dinâmicos com JavaScript

dos dados, assim podemos associar um design diferente para cada barra, por exemplo. Veja como ficaria nosso método `criarGrafico()` ao fazer uso dessa função (**Listagem 5**).

O resultado pode ser visualizado na **Figura 6**. Vamos aos detalhes:

- Na linha 2 vemos a utilização da função `arrayToDataTable()` para efetuar a conversão do vetor de dados passado como parâmetro para um objeto do tipo `DataTable`.

Listagem 5. Refatorando função `criarGrafico()`.

```
01 function criarGrafico() {
02     var data = google.visualization.arrayToDataTable([
03         ['Dia', 'Atividades Completas', 'Atividades Pendentes', { role:'style' },
04         { role:'annotation'}],
05         [{v: [8, 0, 0], f:'8 AM'}, 2, 8.75, 'color: red;\'8 AM'],
06         [{v: [9, 0, 0], f:'9 AM'}, 2.5, 7.5, 'color: blue; 2.5],
07         [{v: [10, 0, 0], f:'10 AM'}, 3.9, 6.3, 'opacity: 0.2;\'Pendentes: 6.3'],
08         [{v: [11, 0, 0], f:'11 AM'}, 4, 5.25, 'stroke-color: black; stroke-width: 1;
09         fill-color: yellow;\'11 AM'],
10         [{v: [12, 0, 0], f:'12 PM'}, 6, 4.25, 'stroke-color: pink; stroke-opacity: 0.6;
11         stroke-width: 1; fill-color: green; fill-opacity: 0.2;?']
12     ]);
13
14     var options = {
15         width: 1500,
16         height: 500,
17         title: 'Balanço de Atividades Entregues/Dia',
18         hAxis: {
19             title: 'Dia',
20             format: 'h:mm a',
21             viewWindow: {
22                 min: [7, 30, 0],
23                 max: [12, 0, 0]
24             }
25         },
26     };
27
28     var chart = new google.visualization.ColumnChart(
29         document.getElementById('chart'));
30
31     chart.draw(data, options);
32 }
```

- Na linha 3 configuramos as duas roles que atuarão diretamente no estilo (`style`) e nas anotações dentro de cada barra (`annotation`), respectivamente. Isso fará com que, aliada aos valores configurados para essas propriedades até a linha 8, cada barra tenha estilo próprio e texto exibido sem que tenhamos de passar o mouse sobre a mesma.
- Esse exemplo reduziu a quantidade de horas para apenas 12hs, mas se o leitor desejar usar o dia inteiro, pode configurar a propriedade `bar` que vimos para aumentar o tamanho das barras.

Ainda sobre gráficos de barras e colunas, podemos criar um subtipo desse gráfico, o gráfico em pilha. Esse tipo de gráfico funciona colocando os valores um acima do outro, em forma de pilha. Se existir qualquer valor negativo, ele será posicionado de igual forma abaixo do eixo X um sob o outro. Para adaptar ao nosso modelo de negócio, suponha que queremos criar um gráfico que exiba a mesma média salarial, só que por ano. Cada pedaço da barra será uma profissão com o respectivo valor de cada salário. Vejamos a **Listagem 6**.

O resultado pode ser visualizado na **Figura 7**. Vejamos os detalhes:

- Na linha 3 temos a mesma implementação da função de conversão de vetor para `DataTable`. A diferença está na ordem dos dados no vetor. Agora temos as descrições dos profissionais como primeiro dado, como se estivéssemos preenchendo uma tabela no Excel. Note que o cabeçalho dessa tabela também deve existir, no caso a label "Profissional".
- A partir da linha 5 injetamos cada um dos valores que devem sempre começar com o ano seguido das variações salariais.
- Na linha 16 temos o atributo novo `isStacked`, que foi configurado com `true` para dizer que nosso gráfico de colunas será exibido em forma de pilha.

Material Design

Em 2014, o Google anunciou orientações destinadas a apoiar uma aparência comum em todas as suas propriedades e aplicativos (como os apps Android) que são executados em plataformas do

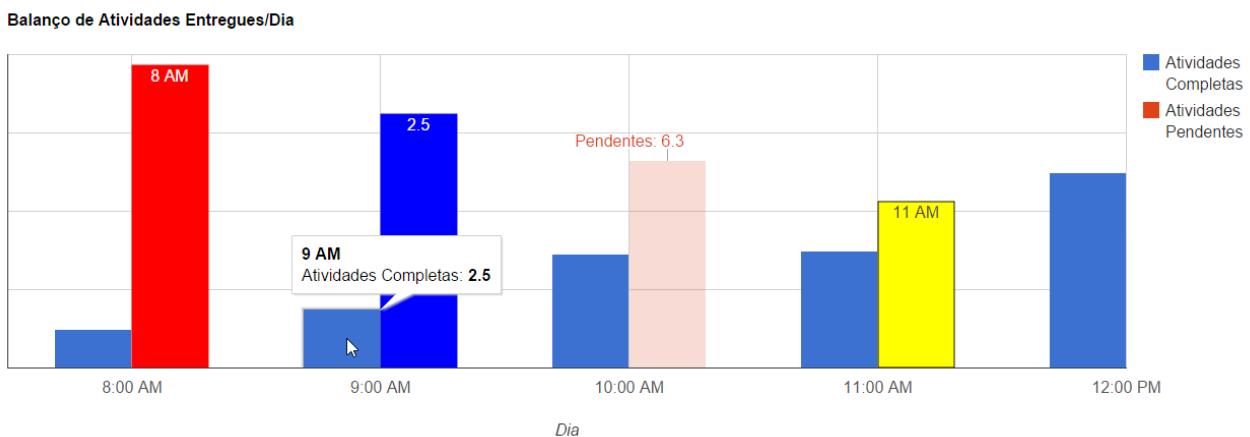


Figura 6. Gráfico de barras usando formatações diferentes.

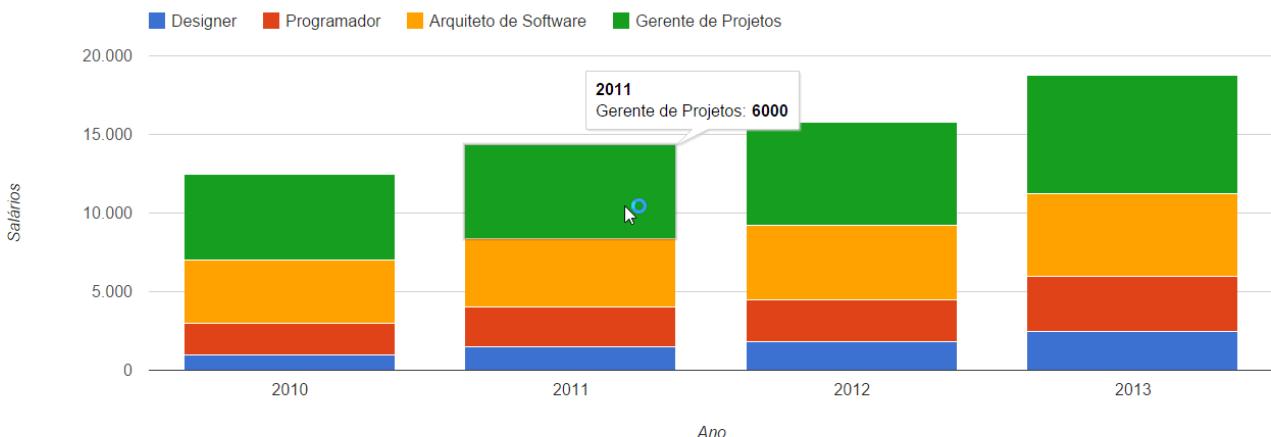


Figura 7. Gráfico em pilha usando mesmo exemplo de profissionais

próprio Google. A esse novo conceito foi dado o nome de *Material Design* (Design de Materiais).

Para criar um gráfico de colunas *Material* basta carregar a API de visualização do Google (agora com o pacote 'bar' em vez do 'corechart' que estávamos usando), definir a sua tabela de dados, e em seguida, criar o objeto de gerenciamento do gráfico (por intermédio da classe google.charts.Bar).

A principal vantagem ao usar esse novo conceito, além de um design mais elegante, é poder mudar facilmente a orientação do gráfico, sem perder em responsividade. Tomando como base o exemplo anterior, a nossa listagem ficaria semelhante à **Listagem 7**.

Listagem 6. Exemplo de gráfico em pilha.

```

01 // função de callback
02 function criarGrafico() {
03   var data = google.visualization.arrayToDataTable([
04     ['Profissional','Designer','Programador','Arquiteto de Software',
05      'Gerente de Projetos'],
06     ['2010',1000,2000,4000,5500],
07     ['2011',1500,2500,4350,6000],
08     ['2012',1800,2700,4750,6500],
09     ['2013',2500,3500,5250,7500]
10   ]);
11   var options = {
12     width: 1500,
13     height: 500,
14     legend: { position: 'top', maxLines: 3 },
15     bar: { groupWidth:'75%' },
16     isStacked: true,
17     hAxis: {
18       title:'Ano'
19     },
20     vAxis: {
21       title:'Salários'
22     }
23   };
24   var chart = new google.visualization.ColumnChart(
25     document.getElementById('chart'));
26   chart.draw(data, options);
27
28
29 }
```

Listagem 7. Exemplo de gráfico usando o conceito do Material Design.

```

01 <script type="text/javascript">
02   /// Carrega a API da biblioteca Visualization e add os nomes dos pacotes.
03   google.load('visualization','1.1',{'packages': ['bar']});
04
05   // seta a função de callback
06   google.setOnLoadCallback (criarGrafico);
07
08   // função de callback
09   function criarGrafico() {
10     var data = google.visualization.arrayToDataTable([
11       ['Ano','Designer','Programador','Arquiteto de Software',
12         'Gerente de Projetos'],
13       ['2010',1000,2000,4000,5500],
14       ['2011',1500,2500,4350,6000],
15       ['2012',1800,2700,4750,6500],
16       ['2013',2500,3500,5250,7500]
17     ]);
18
19   var options = {
20     chart: {
21       title:'Distribuição de profissionais/salários',
22       subtitle:'Sales, Expenses, and Profit: 2014-2017'
23     },
24     series: {
25       0: { axis:'distance' } // Vincula a série de 0 a um eixo chamado "distance".
26     },
27     width: 800,
28     height: 400,
29     legend: { position:'top' },
30     bar: { groupWidth:'75%' },
31     axes: {
32       y: {
33         distance: {label:'Salários'}
34       }
35     }
36
37   var chart = new google.charts.Bar(document.getElementById('chart'));
38   chart.draw(data, options);
39
40 </script>
```

Google Charts: Criando Gráficos dinâmicos com JavaScript

Vamos aos detalhes:

- Na linha 3 mudamos o valor da versão do pacote para 1.1, em vista desse tipo de recurso ser recente e só estar contido neste módulo. Além disso, mudamos também o package para `bar`.
- Na linha 11 mudamos o valor inicial de “Profissional” para “Ano”, uma vez que o Material Design não considera o preenchimento em forma de tabela e sim de barras.
- Na linha 18 temos algumas mudanças quanto à propriedade `options`:

- O título e subtítulo agora precisam estar dentro de um outro atributo: `chart`. Perceba que quando se muda o módulo utilizado todo o código importado também muda, logo precisamos adaptar aos padrões.

- O atributo `series` da linha 23 cria o elemento que se encarregará de configurar os subatributos dos eixos X e Y no plano.

- Na linha 30, o atributo `axes` se encarrega de configurar o eixo Y através do subatributo `y`. As labels agora são definidas dentro desse atributo.

Nota

O Material Design não irá funcionar em versões antigas do Internet Explorer. Do IE8 para trás ele também não suporta o SVG, que é requerido pelo Google Charts.

Outros gráficos

O Google Charts provê uma gama de outros gráficos que podem facilmente se adaptar à realidade dos nossos exemplos. Vejamos alguns exemplos deles.

Gráfico de Linhas

O gráfico de linhas é bem semelhante ao de colunas no que confere o cruzamento das informações, porém em vez de barras ele usa linhas para marcar os pontos. Suponha que no nosso exemplo queremos criar um gráfico cruzando o desempenho dos profissionais da empresa em detrimento de um intervalo de dias. O código para isso se encontra na [Listagem 8](#).

Vejamos algumas observações:

- Na linha 2 temos a mudança do package para o valor “line”. Veja que também estamos usando a versão 1.1, esta que caracteriza sempre o uso do Material Design.
- Na linha 6 voltamos a criar o objeto `DataTable`. Isso porque esse gráfico exige que as linhas sejam adicionadas em forma de tabela, diferente de como fizemos com o preenchimento de barras.
- Na linha 14 populamos o vetor de dados, porém com o primeiro valor como string. Se você precisar usar o dado numérico para manipular algo no JavaScript pode usar as variáveis `v` e `f` que vimos, ou concatenar os valores na mão mesmo.
- Na linha 32 configuramos os títulos. Veja que estamos considerando os valores em DPUs, que é uma unidade muito comum para medir atividades e desempenho.
- Na linha 40 mudamos o tipo do objeto instanciado para `google.charts.Line()`.

O resultado da execução pode ser visualizado na [Figura 10](#).

Gráficos geográficos

Um geochart é nada mais que um mapa de um país, um continente ou uma região com áreas identificadas de três modos:

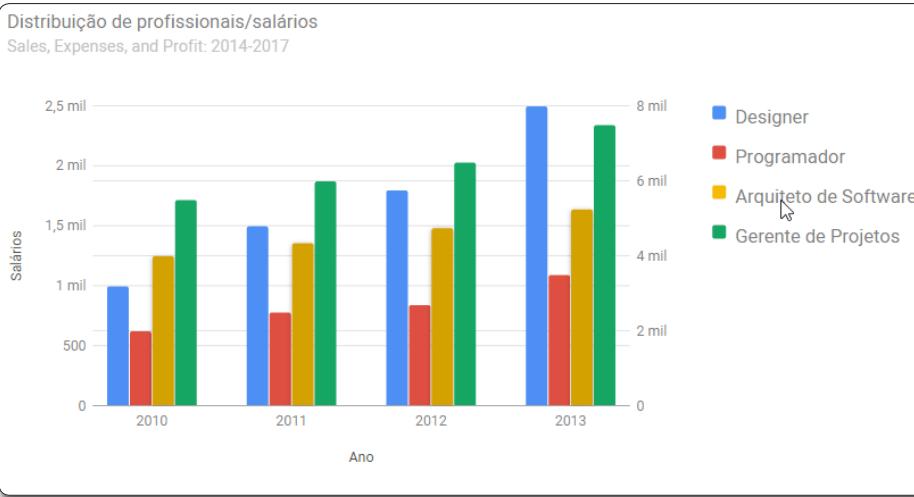


Figura 8. Gráfico em colunas usando Material Design com grupo selecionado

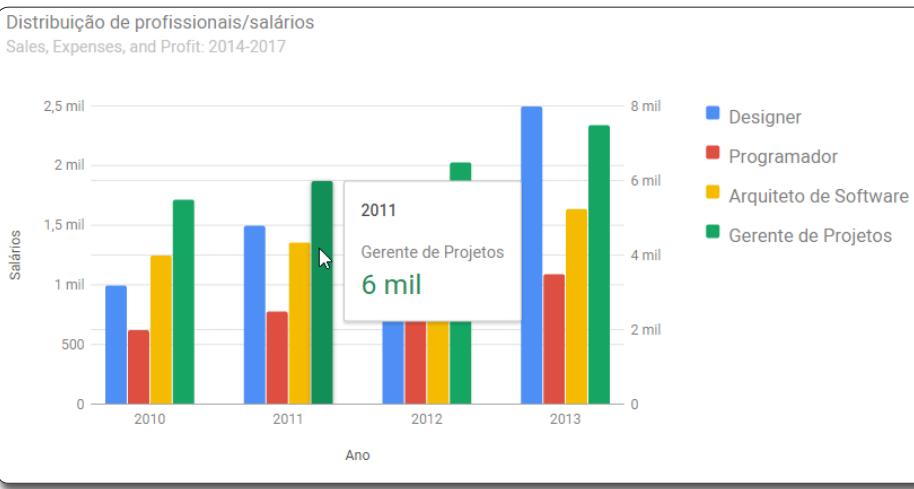


Figura 9. Gráfico em colunas usando Material Design com barra selecionada

Listagem 8. Exemplo de gráfico de linhas usando o Material Design.

```
01 <script type="text/javascript">
02   google.load('visualization','1.1';{packages: ['line']});
03   google.setOnLoadCallback(drawChart);
04
05   function drawChart() {
06     var data = new google.visualization.DataTable();
07
08     data.addColumn('string','Dia');
09     data.addColumn('number','Designer');
10    data.addColumn('number','Programador');
11    data.addColumn('number','Arquiteto de Software');
12    data.addColumn('number','Gerente de Projetos');
13
14    data.addRows([
15      ['Dia 1', 35.8, 80.8, 41.8, 55.5],
16      ['Dia 2', 29.9, 49.5, 32.4, 55.5],
17      ['Dia 3', 24.4, 57, 25.7, 55.5],
18      ['Dia 4', 10.7, 18.8, 10.5, 55.5],
19      ['Dia 5', 10.9, 17.6, 10.4, 50.5],
20      ['Dia 6', 7.8, 13.6, 7.7, 55.5],
21      ['Dia 7', 6.6, 12.3, 9.6, 55.5],
22      ['Dia 8', 11.3, 19.2, 10.6, 15.5],
23      ['Dia 9', 15.9, 42.9, 14.8, 55.5],
24      ['Dia 10', 11.8, 10.9, 11.6, 55.5],
25      ['Dia 11', 4.3, 7.9, 4.7, 55.5],
26      ['Dia 12', 5.6, 8.4, 5.2, 55.5],
27      ['Dia 13', 3.8, 6.3, 3.6, 55.5],
28      ['Dia 14', 4.2, 6.2, 3.4, 55.5]
29    ]);
30
31    var options = {
32      chart: {
33        title: 'Desempenho profissionais/Dia',
34        subtitle: 'em DPUs'
35      },
36      width: 800,
37      height: 450
38    };
39
40    var chart = new google.charts.Line(document.getElementById('chart'));
41
42    chart.draw(data, options);
43  }
44 </script>
```

- O modo **region** (região): onde temos regiões delimitadas por cores, como países, províncias, estados, etc.
- O modo **markers** (marcadores): onde usamos círculos para designar regiões que são dimensionados de acordo com um valor que você especificar.
- O modo **text** (texto): onde rotulamos as regiões com identificadores em texto (por exemplo, "Brasil" ou "América do Sul").

Suponha que agora você precisa gerar um gráfico do Brasil cruzando a quantidade de profissionais por estado do país. Veja o código da **Listagem 9** para isso.

Vejamos os detalhes da listagem:

- Na linha 2 fizemos a substituição do package para **geochart**, e a versão para 1 (ainda não temos geocharts com Material Design).
- Na linha 7 convertemos o vetor para **DataTable**, passando como valores os nomes das cidades. Esses nomes precisam estar corretos, pois o geochart irá checar na base de dados do Google Maps se eles existem.
- Na linha 14, no objeto **options**, temos definidos dois novos atributos:
 - **region**: deverá sempre ser preenchido com a abreviação da região ou o número referente às coordenadas geográficas (Por ex: 'BR' para Brasil, 'US' para Estados Unidos, '002' para África. Você encontrará uma lista completa dos códigos na página do Google Charts).
 - **displayMode**: serve para definir um dos três tipos de gráfico possível. Os valores validos são: 'markers', 'text' e 'region' (este último é o valor default caso nenhum seja informado).

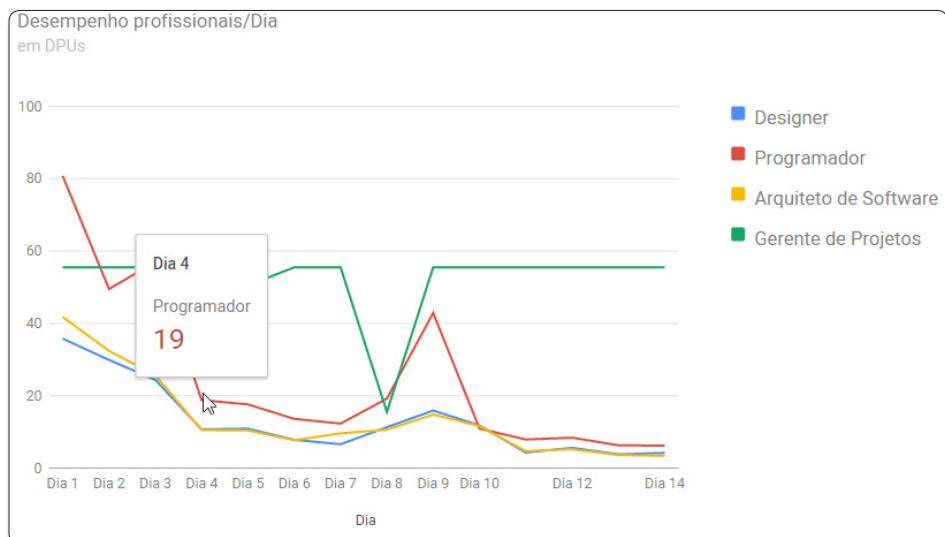


Figura 10. Gráfico em linhas usando Material Design com linha selecionada

- Na linha 21 temos a mudança para a instanciação do objeto de tipo **google.visualization.GeoChart()**.

O resultado pode ser verificado na **Figura 11**.

Note que a barra de proporções exibida na parte inferior à esquerda do gráfico se auto ajustará à medida que os valores aumentarem ou diminuírem.

Essa não é a única forma de distribuir os estados. Também podemos dividir o mapa por todos os estados e fazer a seleção por fronteiras. A única alteração necessária é modificar o valor do **displayMode** para 'region' e incluir o seguinte atributo ao objeto **options**:

```
resolution:"provinces",
```

Google Charts: Criando Gráficos dinâmicos com JavaScript

Listagem 9. Código para geochart que cruza quantidade de profissionais por estado.

```
01 <script type="text/javascript">
02   google.load("visualization", "1", {packages:["geochart"]});
03   google.setOnLoadCallback(desenharMapaRegioes);
04
05   function desenharMapaRegioes () {
06
07     var data = google.visualization.arrayToDataTable([
08       ['Cidade', 'N. Profissionais'],
09       ['São Paulo', 122000],
10      ['Rio de Janeiro', 85000],
11      ['Pernambuco', 60000]
12     ]);
13
14     var options = {
15       region:'BR',
16       width: 800,
17       displayMode:'markers',
18       height: 450
19     };
20
21     var chart = new google.visualization.GeoChart(document.getElementById('chart'));
22
23     chart.draw(data, options);
24   }
25 </script>
```

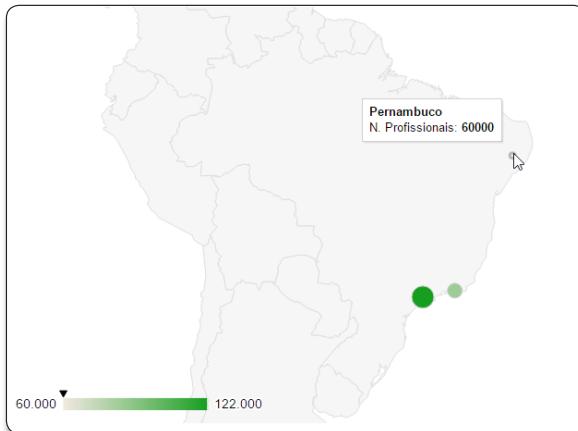


Figura 11. Geochart com distribuição de profissionais por estado

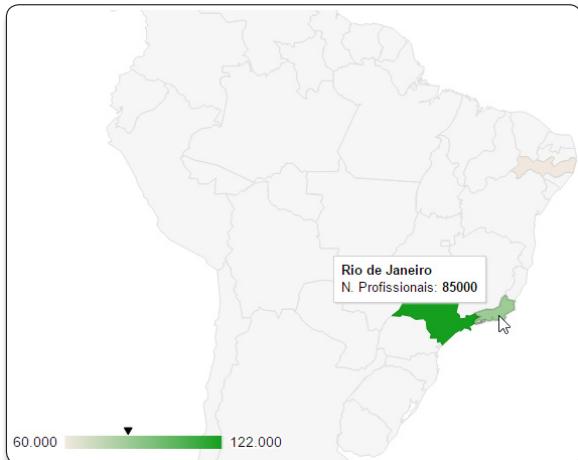


Figura 12. Geochart com distribuição de profissionais via região

O Google Charts entende que estados são províncias e, portanto, se faz necessário a inclusão da palavra-chave. Veja na **Figura 12** o resultado dessa alteração.

Gráficos de timeline

O conceito de **timeline** ficou famoso após o advento das redes sociais. Aplicativos como Facebook, Instagram, etc. tem como principal recurso a exibição das postagens, por exemplo, em uma timeline. O conceito está totalmente relacionado ao tempo: traça-se uma linha do tempo (em anos, meses, etc.) e os eventos são impressos nela.

No nosso exemplo, imagine que queiramos rastrear o tempo que os funcionários de uma empresa estão trabalhando, desde a sua contratação até as demissões. Vejamos o código da **Listagem 10**.

Listagem 10. Código para criar timeline de tempo de trabalho dos funcionários.

```
01 <script type="text/javascript">
02   google.load("visualization", "1.1", {packages:["timeline"]});
03   google.setOnLoadCallback(desenharGrafico);
04
05   function desenharGrafico () {
06     var container = document.getElementById('chart');
07     var chart = new google.visualization.Timeline(container);
08     var dataTable = new google.visualization.DataTable();
09
10     dataTable.addColumn({ type:'string', id:'Id' });
11     dataTable.addColumn({ type:'string', id:'Nome' });
12     dataTable.addColumn({ type:'date', id:'Início' });
13     dataTable.addColumn({ type:'date', id:'Fim' });
14     dataTable.addRows([
15       ['Período','Designer: João Pereira', new Date(2010, 1, 15),
16        new Date(2011, 3, 14)],
17       ['Período','Arquiteto: Maria Beltrão', new Date(2011, 2, 3),
18        new Date(2013, 1, 23)],
19       ['Período','Gerente: Souza Rodrigues', new Date(2010, 5, 30),
20        new Date(2015, 2, 3)]];
21
22     var options = {
23       avoidOverlappingGridLines: false,
24       width: 1000,
25       height: 450
26     };
27     chart.draw(dataTable, options);
28   }
29 </script>
```

Vejamos os detalhes:

- Na linha 2 configuramos o package para ‘timeline’ e a versão para 1.1 para o Material Design.
- Para criar uma timeline precisaremos tanto do objeto DataTable quanto de um objeto do tipo google.visualization.Timeline (linha 7), no qual setaremos os dados e o objeto options.
- Na linha 10 temos a adição das colunas. Note que a primeira é o identificador do período que deverá ser obrigatoriamente uma string.
- Na linha 14 preenchemos as linhas da tabela. Observe que estamos passando agora como parâmetro objetos Date do JavaScript, com o ano, mês e dia passados por parâmetro, respectivamente (padrão americano).

Listagem 11. Migrando gráfico de linhas para de área.

```

01 <script type="text/javascript">
02   google.load('visualization','1.1';{packages: ['corechart']});
03   google.setOnLoadCallback(drawChart);
04
05   function drawChart() {
06     var data = new google.visualization.DataTable();
07
08     data.addColumn('string', 'Dia');
09     data.addColumn('number', 'Designer');
10    data.addColumn('number', 'Programador');
11    data.addColumn('number', 'Arquiteto de Software');
12    data.addColumn('number', 'Gerente de Projetos');
13
14    data.addRows([
15      ['Dia 1', 35.8, 80.8, 41.8, 55.5],
16      ['Dia 2', 29.9, 49.5, 32.4, 55.5],
17      ['Dia 3', 24.4, 57, 25.7, 55.5],
18      ['Dia 4', 10.7, 18.8, 10.5, 55.5],
19      ['Dia 5', 10.9, 17.6, 10.4, 50.5],
20      ['Dia 6', 7.8, 13.6, 7.7, 55.5],
21      ['Dia 7', 6.6, 12.3, 9.6, 55.5],
22      ['Dia 8', 11.3, 19.2, 10.6, 15.5],
23      ['Dia 9', 15.9, 42.9, 14.8, 55.5],
24    ]);
25
26    var options = {
27      title: 'Desempenho profissionais/Dia',
28      legend: {position: 'bottom'},
29      width: 800,
30      height: 450,
31      hAxis: {title: 'Dias', titleTextStyle: {color: 'black'}},
32      vAxis: {title: 'PDUs', minValue: 0}
33    };
34
35    var chart = new google.visualization.AreaChart(
36      document.getElementById('chart'));
37
38    chart.draw(data, options);
39
40  }
41
42  chart.draw(data, options);
43
44 </script>

```

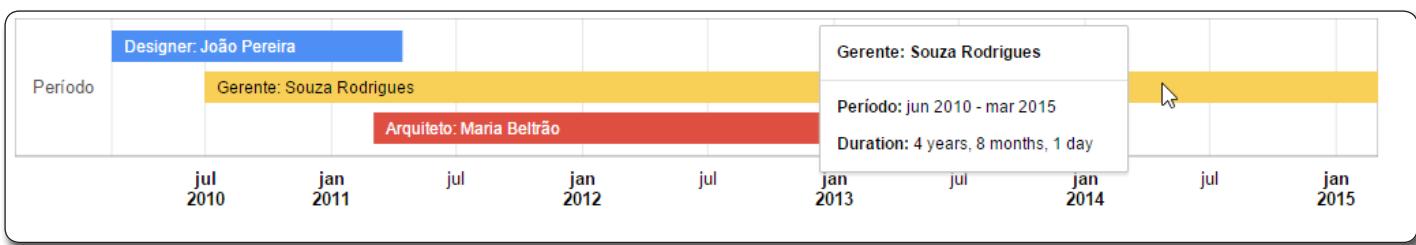


Figura 13. Gráfico em formato timeline

- Na linha 20, dentro do objeto **options**, configuramos o atributo **avoidOverlappingGridLines** como false, que evitará a sobreposição das linhas da grid, ou seja, todas devem ser exibidas uma abaixo da outra.

O resultado pode ser verificado na **Figura 13**.

- Na linha 33 mudamos a posição da legenda para você ver como a API se comporta exibindo-a na parte inferior do gráfico.
- Os títulos para os eixos X e Y devem vir dentro dos objetos **hAxis** e **vAxis**, respectivamente (linhas 36 e 37).
- Na linha 40 instanciamos um novo objeto do tipo **google.visualization.AreaChart()** para representar o gráfico de área.

O resultado da execução pode ser observado na **Figura 14**.

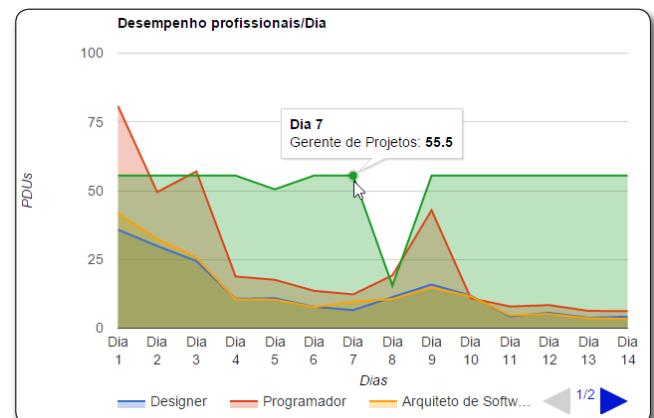


Figura 14. Gráfico de área usando exemplo do de linha

Google Charts: Criando Gráficos dinâmicos com JavaScript

Perceba que o Google Charts gera uma paginação para a legenda quando ele não consegue exibir todos os itens ao mesmo tempo. A propriedade maxLines pode ser configurada para permitir que as legendas ocupem mais que uma linha quando necessário. Além disso, se você passar o mouse sobre cada um dos itens da legenda verá que a área correspondente se destaca em cor e sombra mais escuras.

Existem muitos outros gráficos que o Google Charts disponibiliza, tais como gráficos em pizza (2D e 3D), gráficos de dispersão, combos, gráficos em bolha, tabelas, candlesticks, dentre outros. O importante é sempre focar na divisão da implementação entre gráficos que usam o Material Design, e os que não usam, setando sempre a versão correta para cada caso.

É possível ainda conectar os seus gráficos a fontes de dados remotas, como Web Services, arquivos CSV ou planilhas Excel convertidas em JSON no Google Drive. Para essa finalidade, frameworks como o jQuery, que disponibilizam funções Ajax de requisição assíncrona, ou o AngularJS que provê inúmeros serviços no front-end, podem te ajudar bastante. Agora é com você e a sua criatividade. 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:

Página oficial do Google Charts.

<http://chart.apis.google.com/>

Página do Google Live Charts Playground.

https://developers.google.com/chart/image/docs/chart_playground

DÊ UM SALTO EM CONHECIMENTO!



Acesse o maior
portal para
desenvolvedores
da América
Latina!



Meteor.js: Construindo aplicações web com Node.js e MongoDB

Saiba como integrar um quiz com o servidor

Prototipagem é uma prática que permite que as aplicações sejam estruturadas conforme modelos de código ou via estruturas de marcação (XML, HTML, etc.), possibilitando assim que os dados sejam customizados em detrimento da fonte de dados. O Meteor, ou MeteorJS, é um framework web JavaScript open-source escrito em Node.js. Ele foi inicialmente introduzido à comunidade em dezembro de 2011 com o nome de Skybreak e foi adquirido pelo *Meteor Development Group* da startup Y Combinator em 2014 com o objetivo de expandir o suporte (já fornecido) a banco de dados.

Através dele conseguimos desenvolver rapidamente código de protótipos para aplicações tanto web rodando em browsers, quanto para dispositivos móveis (Android, iOS, Windows Phone, etc.). Com ele, podemos também facilmente integrar a aplicação com o MongoDB (Banco de dados NoSQL baseado em documentos JSON) e usar o protocolo DDP (*Distributed Data Protocol*) para propagar as mudanças nos dados para todos os clientes do serviço em tempo real sem requerer qualquer código de sincronização específico.

Neste artigo trataremos de esclarecer os principais tópicos que envolvem esse framework por meio da construção de um aplicativo de quiz de perguntas: o usuário terá acesso a uma tela de cadastro de perguntas e respectivas opções, assim como a lista de perguntas já criadas irá aparecer abaixo do formulário. O usuário também terá uma pré-lista de perguntas criada quando ele acessar a aplicação a primeira vez, já que ainda não teremos dados populados. As informações serão armazenadas simultaneamente no banco de dados tanto do lado cliente quanto do servidor e qualquer alteração em um ou outro será automaticamente impressa no browser, independente do cliente a acessar (Firefox, Chrome,

Fique por dentro

Este artigo tem como principal objetivo demonstrar o uso do Meteor.js para a construção de aplicações web que rodem tanto no cliente quanto no servidor. Criaremos um aplicativo de quiz dinâmico de perguntas, que pode ser facilmente adaptado para outras ferramentas de pesquisa de opinião, testes e provas online, etc. Neste, o usuário poderá criar novos tópicos e assimilar opções a cada um, tendo os dados sincronizados em tempo real tanto no conteúdo web quanto para o banco de dados.

O Meteor.js flexibiliza essa implementação ao possibilitar que o código tenha o deploy feito de forma automática, isto é, sempre que salvarmos qualquer arquivo no projeto as alterações se espelharão para todos os clientes navegadores sem a necessidade de reload da página. Além disso, aprenderemos a lidar com os packages do framework, que são recursos para importação de bibliotecas JavaScript de terceiros, como o Bootstrap que usaremos para o estilo da aplicação.

WebView, etc.). No final faremos uma tela de login com o *Open Auth* utilizando a API do Facebook para isso. O usuário, portanto, será direcionado para a tela de login e, uma vez logado com sucesso, será redirecionado para a tela do quiz.

O que é o Meteor?

O conceito de Meteor vai além de um simples framework, ele funciona como uma plataforma completa que envolve tecnologias como linguagem de programação, banco de dados e serviços web. Em comparação com o AngularJS, por exemplo, que é um framework amplamente usado para integrar serviços e regras de negócios no lado front-end da aplicação, o Meteor vai além e consegue lidar com a mesma lógica tanto no lado cliente quanto no servidor do sistema. Veja na **Figura 1** uma representação gráfica de como o framework é dividido bem como os tipos de ferramentas

Meteor.js: Construindo aplicações web com Node.js e MongoDB

que ele suporta. Perceba que os itens marcados em negrito representam as tecnologias que o Meteor já traz consigo por padrão, ao passo que os demais tratam-se de itens também aceitos mas que precisam ser baixados e instalados no ambiente.

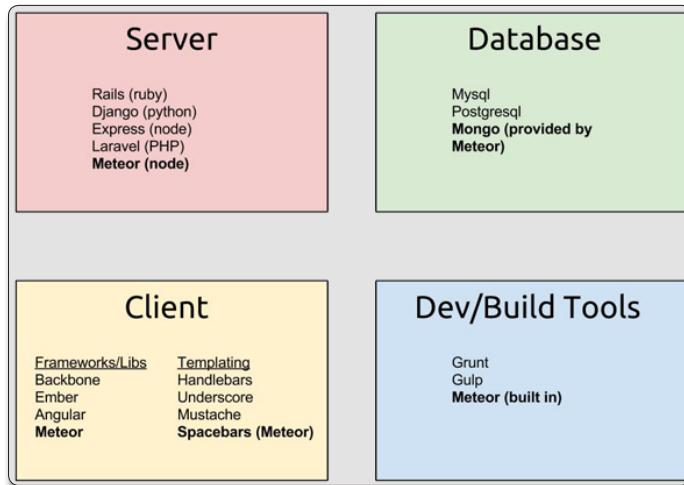


Figura 1. Divisão dos componentes e dependências do Meteor

O Meteor é baseado em sete princípios, a saber:

- Dados na rede:** O Meteor não envia qualquer HTML pela rede, em vez disso ele envia os dados e deixa que o cliente os renderize. Assim, ganhamos em performance ao trafegar bem menos informação, tal como teríamos com os Web Services baseados em restful e JSON, por exemplo.
- Uma só linguagem:** O Meteor possibilita que você escreva ambas as partes (cliente e servidor) da sua aplicação em JavaScript. No lado cliente isso já é padrão por causa do browser e do ECMAScript, no servidor o Node.js entra na jogada para habilitar essa funcionalidade.
- Banco de dados em todo logar:** Você também pode usar os mesmos meios para acessar o banco de dados tanto do lado cliente quanto do servidor. Esse tipo de estratégia difere da que vemos em aplicações que usam serviços para conectar com o banco, gerando requisições HTTP desnecessárias.
- Compensação de latência:** No lado cliente, o Meteor pré-carrega alguns dados e simula os modelos de dados que temos no servidor para fazer com que as chamadas ao servidor pareçam instantâneas. Funciona semelhante aos recursos de cache que temos em outros frameworks server side como c3p0 do Hibernate, por exemplo.
- Update automático das camadas:** No Meteor tempo real é o padrão. Todas as camadas, desde o banco de dados até os templates, se atualizam automaticamente quando necessário. É como se fosse um “hot deploy” por código e para camadas.
- Simplicidade igual a Produtividade:** O Meteor tem uma API extremamente limpa, com códigos e chamadas de métodos simples e usando todos os padrões de qualidade de código Java. Consequentemente, a produtividade aumenta ao não ter de configurar tudo do zero na aplicação.

Além disso, o Meteor também lida com os conceitos de empacotamento dos binários do projeto via atmosphere.js ou via npm do Node.js. Você também pode usar os pacotes do Cordova/PhoneGap para realizar esse trabalho. Não é mais preciso, por exemplo, configurar o Gulp para fluxos de controle e automação diretamente no código, o Meteor já tem o próprio recurso interno para manter todas as dependências alinhadas e sincronizadas. E para conectar com serviços e APIs externos você pode fazer uso do protocolo DDP que citamos, que funciona como um gerenciador de updates: sempre que uma mudança acontecer no servidor sua aplicação será notificada automaticamente.

Meteor e o JavaScript

A principal linguagem de programação usada para desenvolver com o Meteor é o JavaScript. Mas você também pode fazer uso de similares como o CoffeeScript (que no fim gera JavaScript), jQuery, etc. Como o Node.js é baseado em JavaScript, se você quiser ter total suporte no lado servidor e cliente é importante usar essa linguagem como padrão.

O Meteor faz uso dos nomes dos diretórios onde estão arquivados os documentos do projeto para diferenciar entre o que deve ser executado no cliente e no servidor. Qualquer coisa que você colocar no diretório “\server” será qualificado para executar no servidor, da mesma forma para a pasta “\client” que será rodado no cliente. Os arquivos presentes nas demais pastas (como “\lib”, por exemplo) serão executados em ambos os lados. É comum encontrarmos situações onde é preciso executar o mesmo código tanto no cliente quanto no servidor, como a definição das coleções de dados ou validação de formulários que precisam ter dupla camada de segurança, por exemplo.

Entretanto, você não está limitado a estes dois diretórios; até mesmo em arquivos que estiverem fora deles, pode-se simplesmente usar o código representado na **Listagem 1** para checar que se trata de um ou outro. Os atributos booleanos `isClient` e `isServer` servem para identificar se estamos executando no cliente ou servidor, respectivamente.

Listagem 1. Método de verificação do diretório atual.

```
if(Meteor.isClient){  
  // execute código cliente  
}  
  
// OR  
  
if(Meteor.isServer){  
  // execute código servidor  
}
```

Já em relação à forma como devemos importar os arquivos JavaScript (e CSS) nas páginas HTML, não necessariamente precisamos fazer isso dentro da tag head. O Meteor seleciona todos eles, minifica-os e os coloca na sua aplicação sozinho. Funciona de forma semelhante ao que o CoffeeScript e o Sass fazem com os arquivos JS e CSS, respectivamente: o Meteor tem um gerenciador de pacotes

semelhante às gems do Ruby, por exemplo, que empacota todo o código com simples comandos.

Veja o comando a seguir:

```
meteor add coffeescript
```

Ao digitá-lo na interface de linha de comando você terá habilitada a tela para digitar quaisquer comandos em CoffeeScript e eles serão convertidos automaticamente para JavaScript e assimilados às suas respectivas páginas HTML. Esse é só um dos pacotes dos mais de quatro mil disponíveis que você pode encontrar na especificação do Meteor (vide seção **Links** para uma lista completa deles).

Sincronização de Dados

O Meteor consegue manter todos os dados da aplicação sincronizados em ambos os lados da arquitetura, bem como as estruturas que os guardam. Por exemplo, as coleções (listas, filas, mapas, etc.) no Meteor funcionam como encapsuladores de dados síncronos, ou seja, se você alterar qualquer dado no lado servidor a mesma estrutura que o mantém no browser será atualizada automaticamente.

Em relação às estruturas de templates funciona da mesma forma. O Meteor usa o sistema de templates Spacebars para isso, que funciona de forma semelhante ao famoso Handlebars, porém com menos manipulação do DOM para manter as coisas mais rápidas (muito semelhante à API React do Facebook que segue o mesmo conceito). Da mesma forma que os arquivos JS e CSS, não é preciso se preocupar em compilar os templates ou sincronizá-los manualmente, o Meteor faz tudo isso para você.

Veja na **Figura 2** uma representação gráfica de como os dados são sincronizados pelo Meteor.

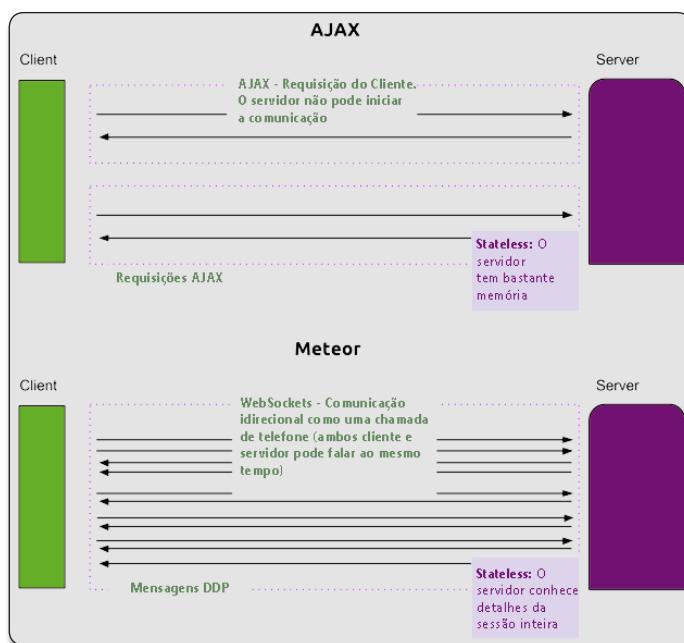


Figura 2. Comunicação entre cliente/servidor no Meteor

O Meteor faz uso tanto dos WebSockets quanto do protocolo DDP para sincronizar os dados. Se fizéssemos uma comparação com o tradicional protocolo HTTP que a maioria das aplicações web usa, teríamos algo como:

- Esse modelo o substitui por completo, criando um novo conceito interno de arquitetura baseado em requisições assíncronas.
- No modelo HTTP o cliente nem sempre sabe o que o servidor está fazendo (como informar ao cliente quando o banco de dados é atualizado) e vice-versa. É uma relação separada, ao contrário do Meteor: tudo que o cliente faz o servidor tem conhecimento, e vice-versa.

Na figura temos ainda uma comparação entre o modelo que faz uso de requisições Ajax (que são amplamente usadas em conjunto com o HTTP para aumentar a performance das aplicações) e o do Meteor. Veja a quantidade de requisições e a velocidade com a qual elas trafegam no segundo gráfico. Esse é o segredo do Meteor: deixar o HTTP (mais lento) de lado e não trafegar HTML pesado, somente dados.

Banco de Dados

No Meteor temos duas representações de banco: O Mongo e o Minimongo. O Mongo fica localizado no servidor e é uma representação verdadeira dos dados, onde de fato serão alocadas as informações perenes. Já o Minimongo é uma biblioteca JavaScript que existe no cliente com a intenção de simular um banco de dados. Geralmente, ele só tem um subconjunto dos dados do Mongo (a menos que você decida publicar todos os dados em um objeto coleção para o cliente), entretanto, independente dos dados que você publique o Meteor irá mantê-los sempre sincronizados: tenham sido eles alterados no cliente ou no servidor.

Veja na **Figura 3** como essa sincronização é feita no Meteor.

Esse tipo de arquitetura se encaixa perfeitamente nas que são usadas para chats. A resposta precisa ser rápida, os dados sincronizados entre cliente-servidor e nenhum HTML é trafegado.

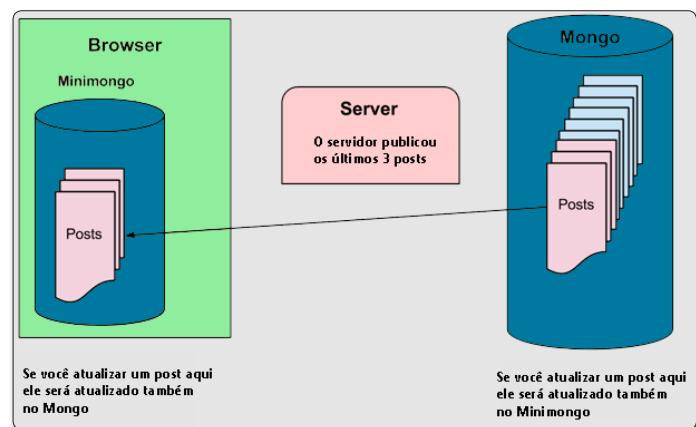


Figura 3. Sincronização de dados via DBs no Meteor

Configuração do ambiente

A instalação do Meteor se dá de duas formas, dependendo do seu Sistema Operacional:

1. Se você usa Linux ou algum dos Sistemas Operacionais baseados em OSX, basta executar a seguinte instrução no terminal de comandos:

```
curl https://install.meteor.com/ | sh
```

2. Caso tenha um Windows então você precisará efetuar o download do instalador específico na página do Meteor (vide seção **Links**).

A primeira opção baixará os arquivos e configurará as variáveis de ambiente automaticamente. Como a maioria dos usuários usa Windows, mostraremos como configurar no mesmo. Após efetuar o download do instalador, execute-o e aguarde até que ele baixe todas as dependências. Quando ele finalizar, uma tela igual à da **Figura 4** irá aparecer solicitando o seu login/senha da conta do Meteor. Você pode optar por criar uma nova conta na janela ou logar com uma já existente, assim seus dados mantêm-se sincronizados em todo ambiente que trabalhar. Se não desejar, basta clicar na opção “*Skip this step*”.



Figura 4. Tela de login/nova conta do Meteor

Após isso, você precisará reiniciar o SO para que as alterações sejam salvas. Faça um teste rápido para ver se tudo ocorreu com sucesso: abra o seu terminal de comandos cmd e digite o seguinte comando:

```
meteor --version
```

Você verá a versão do Meteor instalada (no formato “Meteor 1.x.x.x”). Agora precisamos selecionar o diretório no qual queremos criar o nosso projeto Meteor; navegue até ele via terminal e digite o seguinte comando:

```
meteor create quiz-devmedia
```

Ao final do processamento você verá impressa no console a mensagem da **Figura 5**.

```
D:\Meteor>meteor create quiz-devmedia
quiz-devmedia: created.

To run your new app:
  cd quiz-devmedia
  meteor

D:\Meteor>
```

Figura 5. Mensagem de projeto Meteor criado com sucesso

Três arquivos também serão criados referentes ao HTML, JS e CSS do projeto. Este último vem vazio, pois nenhum estilo foi selecionado ainda para a aplicação ficando a seu cargo definir. Veja o código dos dois primeiros arquivos nas **Listagens 2 e 3**, respectivamente.

Listagem 2. Código da página quiz-devmedia.html.

```
01 <head>
02   <title>quiz-devmedia</title>
03 </head>
04
05 <body>
06   <h1>Welcome to Meteor!</h1>
07
08   {{> hello}}
09 </body>
10
11 <template name="hello">
12   <button>Click Me</button>
13   <p>You've pressed the button {{counter}} times.</p>
14 </template>
```

Listagem 3. Código JavaScript do arquivo quiz-devmedia.js.

```
01 if (Meteor.isClient) {
02   // counter starts at 0
03   Session.setDefault('counter', 0);
04
05   Template.hello.helpers({
06     counter: function () {
07       return Session.get('counter');
08     }
09   });
10
11   Template.hello.events({
12     'click button': function () {
13       // increment the counter when button is clicked
14       Session.set('counter', Session.get('counter') + 1);
15     }
16   });
17
18
19 if (Meteor.isServer) {
20   Meteor.startup(function () {
21     // code to run on server at startup
22   });
23 }
```

Perceba que na tag head (linha 1 da **Listagem 2**) não definimos nenhuma tag script de importação do arquivo de JavaScript que sabemos que o projeto faz uso. Tudo acontece por associação, o Meteor carregará sozinho o arquivo JS e analisará o que deve ser executado no cliente.

Na linha 8 definimos o “uso do template” que foi criado na linha 11. O template deve ter sempre um *name* e o seu conteúdo (botão e parágrafo) nunca irá aparecer, exceto se você explicitar isso através dos operadores dupla-chave {{}} no corpo da página. Esse tipo de seleção é comum em vários frameworks JavaScript, como o CoffeeScript, por exemplo.

Em relação ao JavaScript, perceba que de cara já temos a verificação de qual código pertence ao cliente e qual ao servidor (isClient/ isServer). Esses valores são preenchidos automaticamente pelo Meteor ao iniciar o servidor.

Na linha 3 da **Listagem 3** adicionamos um elemento da HTML ao escopo de sessão do Meteor. O conceito é semelhante aos atributos que temos nas linguagens de programação server side (como ASP .NET ou Java EE) onde cada atributo é sempre constituído pelo par *nome*-*valor*.

Na linha 5 definimos as funções *helpers* que são funções utilitárias de contexto. A única função que temos é counter() que irá retornar o valor da variável de sessão (o Meteor trabalha com escopos, a sessão é um dos mais abrangentes; vamos falar mais sobre eles adiante). Na linha 11 criamos os eventos, semelhante à forma como mapeamos os eventos no jQuery: através de seletores. A string ‘click button’ diz que a função seguinte deve ser executada sempre que um clique for efetuado em qualquer input do tipo ‘button’ na página. O código da função, por sua vez, apenas incrementa em um o valor total do contador e o salva novamente na sessão. A função set() da classe Session sempre sobrescreve o valor se ele já existir, impedindo assim que tenhamos valores duplicados na memória.

Na linha 19 temos o código que roda no servidor. Como nenhuma implementação vem por padrão para este escopo, a função está em branco.

Para executar a aplicação não basta simplesmente abrir o arquivo HTML num browser, precisamos iniciar o servidor Node.js e subir a aplicação no mesmo. Para isso, no terminal de comandos, adentre a pasta do projeto (via comand cd) e execute o comando meteor. Após isso, o Meteor vai subir as instâncias do Node.js, um proxy interno e iniciar a aplicação, mostrando ao final em qual endereço e porta você poderá acessar (geralmente http://localhost:3000/). Um cursor ficará piscando informando que o servidor está levantado, para pará-lo basta digitar Ctrl-C.

Acesse o endereço informado em localhost e você verá a tela ilustrada na **Figura 6**.

Qualquer alteração que fizer nos arquivos do projeto não precisa reiniciar o servidor, ele fará o deploy automático das mudanças e atualizará a página sozinho. Faça um teste: modifique o texto dos títulos da página e verifique a mensagem “Client modified – refreshing” que aparecerá no terminal, bem como a página atualizada.



Figura 6. Tela do aplicativo inicial do Meteor

Construção da aplicação

O Meteor também possibilita a construção da sua aplicação diferenciando código de cliente e de servidor via pastas. Ao criar uma pasta \client e outra \server ele automaticamente direcionará a execução para os respectivos ambientes e não mais teremos de usar os atributos isClient/isServer. Portanto, vamos estruturar os nossos diretórios conforme demonstrado na **Listagem 4**.

Listagem 4. Estrutura de diretórios do projeto.

```
.meteor
| client/           // todo o código do cliente vem aqui
|   |--- components/ // pasta que conterá cada componente do projeto
|   |   |--- quiz-devmedia-form.css
|   |   |--- quiz-devmedia-form.html
|   |   |--- quiz-devmedia-form.js
|   |   |--- quiz-devmedia.css
|   |   |--- quiz-devmedia.html
|   |   |--- quiz-devmedia.js
|   |--- app.body.html // layout para a aplicação inteira
|   |--- app.head.html // layout do cabeçalho das páginas
|   |--- app.js        // js global da app
|   |--- app.css       // css global da app
| collections/      // salvaremos os modelos do mongo
|   |--- quiz.js       // coleções mongo
| server/           // código para o servidor
|   |--- bootstrap.js // datas de exemplo para iniciar a aplicação
```

O primeiro passo para construir a aplicação é criar uma coleção de objetos no mongo. Para isso, abra o arquivo quiz.js na pasta \ collections e adicione o seguinte conteúdo ao mesmo:

```
Quiz = new Mongo.Collection('quiz');
```

Isso é o suficiente para a coleção de que precisamos. O motivo de termos criado esse arquivo na pasta \collections foi porque precisaremos dele em ambos cliente e servidor. Agora vamos criar alguns dados de exemplo para quando nossa aplicação for iniciada. Veja na **Listagem 5** o conteúdo que deverá ser adicionado ao arquivo bootstrap.js.

Note que na linha 2 estamos usando a função do Meteor startup() que será a primeira a ser executada quando o projeto subir no servidor. Na linha 5 verificamos se já existe algum dado de quiz através do método find() do MongoDB disponível através do objeto global Quiz que criamos antes. Este método lista todos os objetos adicionados ao banco e o método count() retorna a quantidade.

Meteor.js: Construindo aplicações web com Node.js e MongoDB

Listagem 5. Código de inicialização do projeto.

```
01 // essa função será executada quando a app for iniciada
02 Meteor.startup(function() {
03
04 // se não tiver nenhum quiz disponível, cria um com dados simples
05 if (Quiz.find().count() === 0) {
06 var quizSimples = [
07 {
08 pergunta: 'Você gostou do Meteor?',
09 alternativas: [
10 { texto: 'Sim, muito!', votos: 0 },
11 { texto: 'Mais ou menos...', votos: 0 },
12 { texto: 'Não. Prefiro JavaScript', votos: 0 }
13 ],
14 },
15 {
16 pergunta: 'Como você avalia este artigo?',
17 alternativas: [
18 { texto: 'ruim', votos: 0 },
19 { texto: 'bom', votos: 0 },
20 { texto: 'excelente', votos: 0 }
21 ],
22 }
23 ];
24
25 // itera sobre todos os quizzes e insere cada um no banco
26 _each(quizSimples, function(quiz) {
27 Quiz.insert(quiz);
28 });
29 }
30});
```

Caso a lista venha vazia criamos um vetor com dois objetos preenchidos (que tem como atributos a pergunta e um outro vetor com as alternativas) para teste. Os valores da quantidade de votos naquela opção serão salvos como subatributos do vetor alternativas.

Na linha 26 iteramos sobre a lista de itens do quiz e inserimos cada um no objeto do Mongo via método insert().

Uma vez que o Meteor implementa uma instância do Mongo no cliente, podemos executar os comandos do MongoDB diretamente do console do browser, via ferramenta do desenvolvedor. Ao ter salvo todos os arquivos, o Meteor já reiniciou a aplicação, portanto os objetos que criamos já estão no banco de dados.

Abra o seu navegador (usaremos o Chrome para os exemplos mostrados) e, dentro da página que executamos no Meteor, acesse o console de depuração. Digite o seguinte comando:

```
Quiz.find().fetch()
```

O resultado será a impressão dos dois objetos inseridos, tal como mostra a **Figura 7**. Observe que existem outros atributos que o Meteor cria para dar suporte ao objeto quiz, como um identificador (`_id`) e o tamanho do vetor (`length`). Isso constitui mais um benefício do Meteor uma vez que os dados estão sempre disponíveis para debugging sem necessitar o uso do operador debugger ou de *break points*.

Nota

Consulte o site oficial do MongoDB na seção [Links](#) para ver a lista de opções disponíveis via linha de comando. Caso queira limpar o banco de dados basta executar o comando `meteor reset` no terminal

Template da aplicação

Precisamos configurar os nossos templates que representarão o cabeçalho e corpo das páginas para que não tenhamos que replicar o código em cada uma delas. Vamos começar pela página `app.html` adicionando o conteúdo da **Listagem 6** à mesma.

A página tem uma composição simples onde, na linha 6, importamos o template do formulário do quiz, que enxertará a HTML dentro da referida div. Nessa linha, o Meteor irá procurar pelo arquivo que tenha a tag `<template name='formQuiz'>`. Note que as classes que estamos criando são relativas ao Bootstrap: organizadas em colunas.

Na linha 12 criamos a div que irá iterar sobre todos os itens do quiz. Veja o `forEach` que usamos na linha 13, até mesmo estruturas de repetição devem estar entre os sinais de dupla chave. E mais uma vez importamos o template do quiz na linha 14. Toda importação deve sempre começar com o sinal de maior “>”, e todo fim de bloco deve finalizar com o nome do comando precedido de uma barra “/”.

Vamos editar agora o arquivo do cabeçalho (**Listagem 7**). Trata-se de um cabeçalho comum a outras páginas HTML.

Se você verificar o código HTML gerado da página carregada no browser perceberá que dentro da tag `head` temos vários arquivos sendo importados (JS e CSS) que guardam toda a lógica autogerada pelo Meteor.

Uma vez com os templates de cabeçalho e rodapé prontos, ainda não podemos executá-los pois precisamos antes criar as páginas de formulário que os usarão por padrão. Abra o arquivo `quiz-devmedia-form.html` e adicione o conteúdo da **Listagem 8**.

```
> Quiz.find().fetch()
< [▼ Object { _id: "ypcm8Qe4DR8GNqRcx", alternativas: Array[3], ... }, ▼ Object { _id: "hqPHdc6H54s9FMJz", alternativas: Array[3], ... } ]
```

The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. The command `Quiz.find().fetch()` was run, and the output is shown as an array of two objects. Each object represents a quiz document from the MongoDB collection. The first quiz has an ID of `"ypcm8Qe4DR8GNqRcx"` and three alternatives: `"Sim, muito!"`, `"Mais ou menos..."`, and `"Não. Prefiro JavaScript"`. The second quiz has an ID of `"hqPHdc6H54s9FMJz"` and three alternatives: `"ruim"`, `"bom"`, and `"excelente"`. The output also includes internal object structures like `__proto__` and `length`.

Figura 7. Resultado da inspeção da lista de quiz

Listagem 6. Código template da página do corpo.

```
01 <body>
02   <!-- Importa o formulário de criação do quiz aqui -->
03   <div class="container">
04     <div class="row">
05       <div class="col-md-6 col-md-offset-3">
06         {{ formQuiz }}
07       </div>
08     </div>
09   </div>
10
11  <!-- Itera sobre os quizzes, exibindo-os -->
12  <div class="quiz">
13    {{ #each quizzes }}
14      {{ >quiz }}
15    {{ /each }}
16  </div>
17 </body>
```

Listagem 7. Código template da página de cabeçalho.

```
01 <head>
02   <meta charset="utf-8">
03   <title>Meu Quiz DevMedia!</title>
04
05   <!-- Todo o conteúdo CSS será carregado aqui automaticamente -->
06 </head>
```

Listagem 8. Código HTML da página de formulário do quiz.

```
01 <template name="formQuiz">
02   <form>
03     <div class="form-group">
04       <label>Pergunta</label>
05       <input type="text" name="pergunta" class="form-control"
06           placeholder="Sua Pergunta">
07     </div>
08
09     <div class="form-group">
10       <label>Opção #1</label>
11       <input type="text" name="opcao1" class="form-control"
12           placeholder="Opção #1">
13     </div>
14
15     <div class="form-group">
16       <label>Opção #2</label>
17       <input type="text" name="opcao2" class="form-control"
18           placeholder="Opção #2">
19     </div>
20
21     <div class="form-group">
22       <label>Opção #3</label>
23       <input type="text" name="opcao3" class="form-control"
24           placeholder="Opção #3">
25     </div>
26
27     <button type="submit" class="btn btn-lg btn-primary btn-block">
28       Criar Quiz</button>
29   </form>
30 </template>
```

Dessa vez temos somente conteúdo HTML. Note que a tag `<template>` marca o conteúdo a ser importado; o nome também tem de ser igual aos usados na importação da página de corpo. Note também que cada campo de input tem configuradas as propriedades `name` (que será usada para recuperar o valor do campo mais à frente), `class` (com o CSS do Bootstrap) e `placeholder` (que contém a dica do campo quando nada tiver sido preenchido).

No momento, o usuário estará limitado a criar somente três opções de escolha por simplicidade, mas você pode adicionar quantas desejar, inclusive criar um mecanismo para adicionar mais campos dinamicamente. Para não deixar a exibição comprometida, vamos adicionar algum CSS ao arquivo `quiz-devmedia-form.css` (**Listagem 9**).

Agora volte novamente à página e veja o resultado (**Figura 8**). Mesmo com as classes do Bootstrap adicionadas, nenhum estilo relacionado ao mesmo foi impresso porque ainda precisamos importá-lo (faremos isso mais à frente).

Vamos criar agora o conteúdo da página `quiz-devmedia.html` de fato. É nela que iremos iterar a lista de perguntas que debugamos e exibi-las. Veja o código contido na **Listagem 10**.

Opção #1	Opção #1
Opção #2	Opção #2
Opção #3	Opção #3

Figura 8. Formulário do Quiz com CSS inicial

Listagem 9. Código CSS para estilo inicial da página de formulário do quiz.

```
/* Seu CSS vem aqui */
.pergunta-group {
  margin-bottom:20px;
  background:#EEE;
  padding:20px;
}
.pergunta-group label {
  font-size:18px;
}
```

Listagem 10. Código da página `quiz-devmedia.html`.

```
01 <template name="quiz">
02   <div class="quiz well well-lg" data-id="{{ _id }}>
03
04     <h3>{{ pergunta }}</h3>
05
06     {{ #each indexedArray alternativas }}
07       <a href="#" class="voto btn btn-primary btn-block" data-id="{{ _index }}>
08         <span class="votos pull-right">{{ votos }}</span>
09         <span class="text">{{ texto }}</span>
10       </a>
11     {{ /each }}
12   </div>
13 </template>
```

Na linha 1 criamos um novo template de nome `quiz` que será importado na página de corpo. Na linha 4 acessamos os valores da propriedade `pergunta` envolvendo em uma tag `h3` de título. Após isso, na linha 6, iteramos sobre a lista de alternativas disponíveis imprimindo-as em forma de link para que o usuário possa clicar e votar na opção; e dentro de cada link exibiremos a quantidade atual de votos e o texto da descrição.

O atributo **data-id** definido na linha 7 com o valor *_index* será responsável por identificar cada questão com um número de sufixo.

Agora precisamos dar à página do corpo acesso ao banco de dados. Para fazer isso precisamos atribuir a variável do Mongo ao template. Veja na **Listagem 11** esse código que deverá estar contido no arquivo app.js.

Esse código disponibiliza a função “quizes” para ser acessada na página de corpo. A sua única função é acessar o banco de dados e retornar a lista de quizzes.

Perceba que também adicionamos o vetor *indexedArray* ao *forEach* das perguntas. Isso foi feito porque não temos acesso direto aos índices quando iteramos sobre listas no Meteor. Esse é um tipo de recurso básico que os desenvolvedores do Meteor estão trabalhando para incluir nas próximas versões. Logo, precisamos criar esse vetor dentro do app.js tal como na **Listagem 12**.

Listagem 11. Código para funções helper no app.js.

```
01 Template.body.helpers({  
02   quizzes: function() {  
03     return Quiz.find();  
04   }  
05});
```

Listagem 12. Código para o vetor indexedArray de iteração.

```
01 // add um índice a cada item  
02 UI.registerHelper('indexedArray', function(contexto, opcoes) {  
03   if (contexto) {  
04     return contexto.map(function(item, indice) {  
05       item._index = indice;  
06       return item;  
07     });  
08   }  
09});
```

Esse trecho de código se comunica diretamente com o atributo **data-id** que criamos antes: ele apenas cria uma espécie de registro para cada índice e todo o trabalho é feito pela função *registerHelper()* interna à API do Meteor. Dessa forma, agora podemos visualizar as opções na tela recarregada mesmo que sem estilo algum (**Figura 9**).

Eventos do Formulário

Com toda a estrutura preparada, precisamos agora adicionar os eventos e variáveis aos templates. Cada arquivo de formulários HTML tem um arquivo JavaScript correspondente. Vamos começar pelo quiz-devmedia-form.js (**Listagem 13**).

A função events (linha 1) deve ser usada sempre que se deseja adicionar eventos a um template. Na linha 3 estamos mais uma vez recuperando o objeto formulário via seletores do Meteor. Na linha 6 impedimos que o usuário efetue mais de um clique no botão de submit.

Na linha 9 criamos uma nova instância do vetor de perguntas recuperando cada um dos valores digitados nos campos do formulário.

Para recuperar o valor de um campo pelo nome basta acessar o objeto *event* e a propriedade *target* com o nome do respectivo campo. Já as propriedades *votos* iniciam vazias.

Figura 9. Formulário do quiz com perguntas exibidas

Listagem 13. Funções de eventos no JS do formulário.

```
01 Template.formQuiz.events({  
02   // lida com o submit do formulário  
03   'submit form': function(event) {  
04     // impede que o formulário envie mais de uma vez  
05     event.preventDefault();  
06     // recupera os dados do form  
07     var novoQuiz = {  
08       pergunta: event.target.pergunta.value,  
09       alternativas: [  
10         { text: event.target.opcao1.value, votos: 0 },  
11         { text: event.target.opcao2.value, votos: 0 },  
12         { text: event.target.opcao3.value, votos: 0 }  
13       ]  
14     };  
15     // cria o novo quiz  
16     Quiz.insert(novoQuiz);  
17   }  
18 };
```

Finalmente, na linha 19 inserimos o atributo no objeto Quiz do Mongo. Mas ainda precisamos adicionar os eventos de votar para quando o usuário clicar no link de cada opção. Portanto, abra o arquivo quiz-devmedia.js e edite-o com o conteúdo da **Listagem 14**.

Na linha 3 mapeamos a seleção para o evento de click da classe CSS “voto” na página. Na linha 6 prevenimos o duplo click novamente e na linha 9 selecionamos o id do objeto pai do quiz que será usado para distinguir entre as opções do quiz filho.

Na linha 15 adicionamos o valor inicial de cada item do quiz, enquanto na linha 18 incrementamos em um a quantidade de votos para aquela opção em específico. O operador **#inc** será responsável por isso.

Listagem 14. Funções de eventos no JS do quiz.

```
01 Template.quiz.events{  
02 // evento para lidar com o click na opção  
03 'click .voto': function(event) {  
04  
05 // impede o click duplo  
06 event.preventDefault();  
07  
08 // recupera o id do quiz pai  
09 var idQuiz = $(event.currentTarget).parent('.quiz').data('id');  
10 var idVoto = $(event.currentTarget).data('id');  
11  
12 // cria o objeto de incremento  
13 var voteString = 'alternativas.' + idVoto + 'votos';  
14 var action = {};  
15 action[voteString] = 1;  
16  
17 // incrementa o número de votos para esta opção  
18 Quiz.update(  
19 { _id: idQuiz },  
20 { $inc: action }  
21 );  
22 }  
23});
```

Agora é só ir até a página e clicar nas opções. Você verá os itens sendo atualizados (**Figura 10**). Se você publicar a aplicação e tiver várias pessoas usando poderá ver os itens atualizando automaticamente em todas as páginas. Faça um teste, abra a sua aplicação em dois browsers distintos e incremente as opções em um observando elas se atualizarem no outro.

Você gostou do Meteor?

[2 Sim, muito!](#) [0 Mais ou menos...](#) [0 Não, Prefiro JavaScript](#)



Como você avalia este artigo?

[0 ruim](#) [0 bom](#) [1 excelente](#)

Figura 10. Formulário do Quiz com opções incrementadas

Estilo com Bootstrap

Vamos adicionar agora o Bootstrap à aplicação para aplicar alguns estilos rápidos. Uma das vantagens ao se trabalhar com o Meteor é que só precisamos instalar os pacotes das bibliotecas de terceiros via linha de comando para que sejam reconhecidas pelo projeto, nada mais. Para importar o Bootstrap de maneira tradicional teríamos de baixar o pacote, descompactar e mover para a pasta do projeto, além de referenciar os arquivos certos nas tags *link* e *script* do cabeçalho e lidar com problemas de versão.

Com o Meteor só precisamos adicionar esse package ao nosso projeto e pronto. Para isso, execute o seguinte comando no terminal de comandos dentro da pasta do projeto:

```
meteor add twbs:bootstrap
```

Aguarde até o processamento terminar e a mensagem de sucesso ser exibida. Agora é só retornar à página e todo o design do projeto terá mudado igual ao que vemos na **Figura 11**.

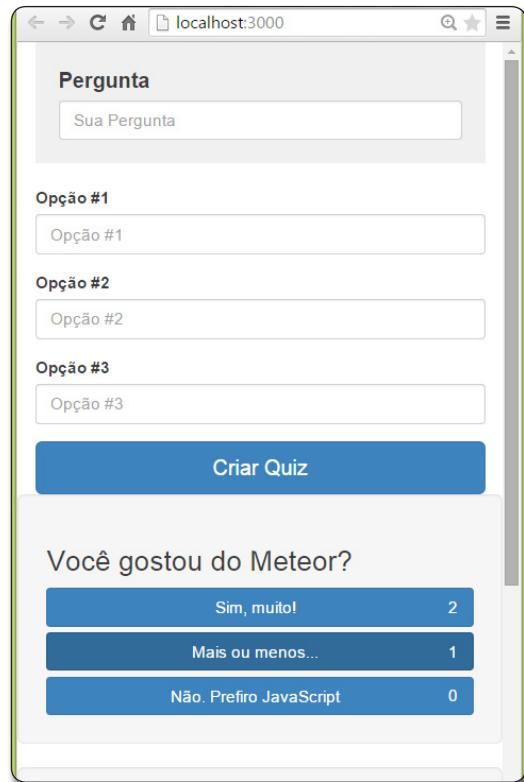


Figura 11. Página do quiz com o Bootstrap

Login OAuth com Facebook

Um dos pacotes que o Meteor disponibiliza dentro da sua especificação é o pacote de autenticação do Facebook. Esse tipo de recurso, conhecido como *Open Authentication*, é muito comum nas aplicações web para facilitar o acesso dos usuários às mesmas sem perder tempo criando um cadastro completo. Os mesmos passos também podem ser seguidos para implementar usando a API do Google Plus e Twitter, entre outras.

O primeiro passo é criar uma aplicação na página de desenvolvedores do Facebook (vide seção **Links**). No menu Apps selecione a opção “Add a New App” e escolha o tipo de aplicação a qual deseja logar: Site. Dê um nome à mesma: *Meteor-Teste*, e clique em *Criar*.

Quando estiver criada, vá até a opção “Test Apps” no menu da lateral esquerda e clique no botão “Create a Test App”. Você será redirecionado para a popup demonstrada na **Figura 12**. Informe um nome para o App de teste e clique em “Create Test App”. Você será levado a uma nova popup para confirmar um captcha.

Após isso, a aplicação de teste estará configurada. Vá até o menu “Settings” e clique no botão “+ Add Platform”, selecionando a opção “Site” novamente. No campo Site URL informe o endereço que estamos testando: <http://localhost:3000> e clique em “Save Changes” (para toda alteração sempre clique no botão de salvamento que fica no fim da página, senão suas configurações serão perdidas).

Agora vá na aba “Advanced” na mesma página e na seção “OAuth Settings”, dentro do campo “Valid OAuth redirect URIs” coloque a seguinte URL: http://localhost:3000/_oauth/facebook?close. Essa URL representa o endereço de redirecionamento para o qual o Facebook deve enviar a resposta à requisição de login.

Com as configurações feitas só precisamos voltar à página de detalhes do app e copiar as informações de “App ID” e “App Secret” que serão usadas pelo nosso código.

Para configurar o acesso no lado do Meteor precisamos primeiro adicionar os pacotes do OAuth e Facebook como dependência ao projeto. Para isso, vá até o terminal de comandos e, de dentro do projeto, execute os seguintes comandos:

```
meteor add accounts-facebook  
meteor add service-configuration
```

Eles criaram os pacotes do login via Facebook e das configurações do mesmo serviço, respectivamente. Aguarde até que eles finalizem, vá até o projeto e crie três novos arquivos: *login.html* e *login.js* no diretório \client e *social-config.js* no diretório \server. Os dois primeiros serão responsáveis por conter os códigos HTML e JavaScript para o login e o último tratará das configurações de login, bem como guardará os dados da autenticação. Vamos começar por este então: abre o arquivo e adicione o conteúdo da **Listagem 15**.

A classe *ServiceConfiguration*, que agora disponível através dos imports que fizemos, faz todo o trabalho de remover o serviço do Facebook (caso ele já exista) e adicionar novamente. Na inserção que fazemos na linha 5 precisamos informar o nome do serviço de autenticação, o id da aplicação que criamos e a senha (*app secret*).

E para exibir a página com o botão de login vamos criar um novo template e adicionar ao arquivo *login.html* que criamos (**Listagem 16**).

O template traz novas configurações que envolvem o uso de serviços e de usuários. O objeto *currentUser* refere-se ao usuário global logado no momento. Dentro dele podemos acessar o objeto *services* que, por sua vez, terá o objeto *facebook* como dependência. Os valores de nome, gênero, etc. devem ser acessados com as

palavras-chave respectivas em inglês. Há cada nova tecnologia OAuth adicionada mais objetos ficam disponíveis dentro do services (Twitter, Google, etc.).

A estruturas condicionais que vemos nas linhas 2 e 6 funcionam como um if-else comum, sendo que o primeiro teste verifica se o usuário é diferente de null, mesmo sem precisar explicitar o teste no código. Assim, montamos dois botões distintos para cada situação: um para login e outro para logout.

Listagem 15. Código de configuração da autenticação com o Facebook.

```
01 ServiceConfiguration.configurations.remove({  
02   service: 'facebook'  
03 });  
04  
05 ServiceConfiguration.configurations.insert({  
06   service: 'facebook',  
07   appId: 'seu_App_Id',  
08   secret: 'sua_Senha_do_App'  
09 });
```

Listagem 16. Código de definição do template de login.

```
01 <template name="login">  
02   {{#if currentUser}}  
03     {{currentUser.services.facebook.name}} -  
04     {{currentUser.services.facebook.gender}}  
05     <button id="logout" class="btn btn-lg btn-primary btn-danger">  
06       Logout</button>  
07   {{else}}  
08     <button id="facebook-login" class="btn btn-lg btn-primary btn-block">  
09       Login com Facebook</button>  
10   {{/if}}  
11 </template>
```

Ainda precisamos definir os eventos de click nestes botões. Portanto, abra o arquivo *login.js* e adicione o conteúdo da **Listagem 17**.

Na linha 2 mapeamos o seletor do click nos campos de id *facebook-login* atribuindo a função conseguinte ao mesmo. A mesma função se encarrega de chamar o método *loginWithFacebook()* da API do Meteor recebendo como resposta o objeto de erro e lançando uma exceção caso algum aconteça. Na linha 10 criamos a função de logout que fará o mesmo processo de chamada à API

lançando uma mensagem de erro caso algo de errado aconteça.

Para finalizar precisamos abrir o arquivo *app.body.html* e modificar a importação do template da linha 6 para >*login* e comentar (ou remover) a div de quiz que irá aparecer junto da outra. Feito isso, podemos ir no browser e verificar o resultado (**Figura 13**).

Após clicar no botão de Login você verá uma nova janela abrir igual à da **Figura 14**. Clique em OK e você será autenticado com sucesso, tendo como resultado a página da **Figura 15**.

Se desejar deslogar da aplicação, basta clicar no botão e você será redirecionado para a

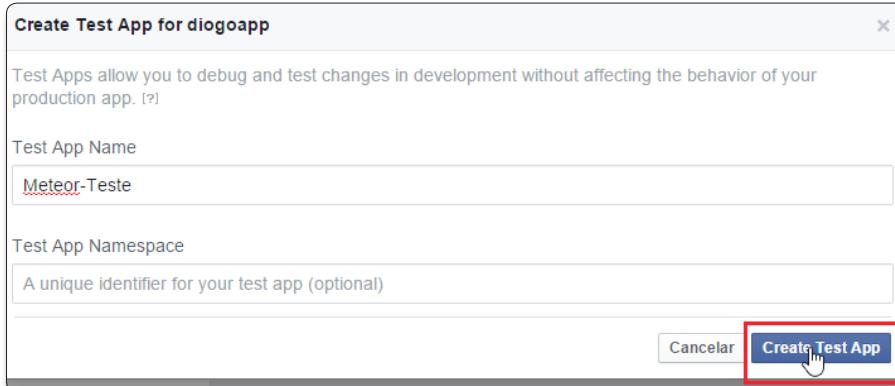


Figura 12. Pop-up para confirmar nome do novo app de teste

Listagem 17. Código de criação dos eventos de click para o template de login.

```
01 Template.login.events({
02   'click #facebook-login': function(event) {
03     Meteor.loginWithFacebook({}, function(err){
04       if (err) {
05         throw new Meteor.Error("Login com Facebook falhou");
06       }
07     });
08   },
09
10  'click #logout': function(event) {
11    Meteor.logout(function(err){
12      if (err) {
13        throw new Meteor.Error("Logout falhou");
14      }
15    })
16  }
17});
```

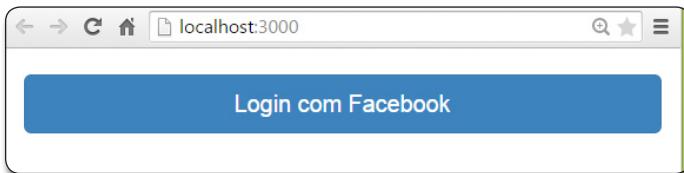


Figura 13. Página de login com o Facebook

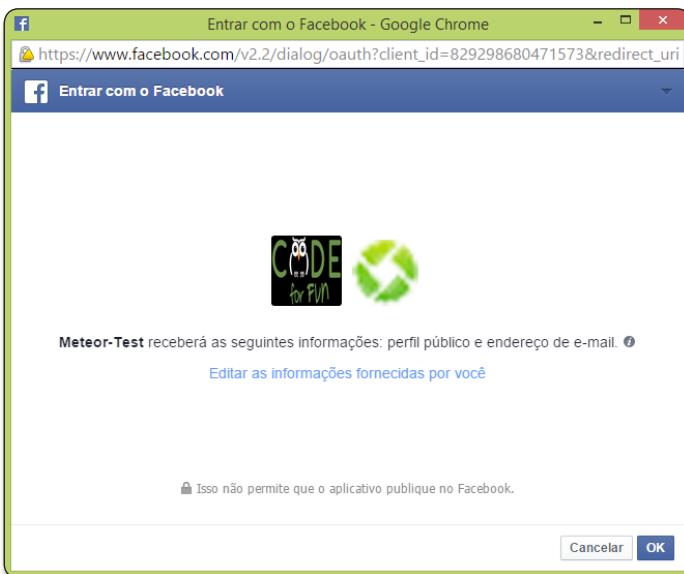


Figura 14. Página de confirmação do login com o Facebook

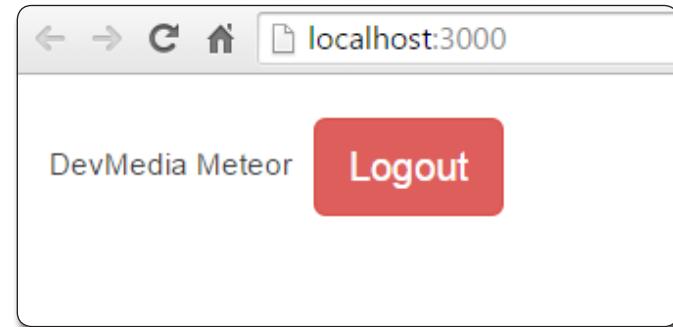


Figura 15. Autenticação com sucesso. Página de Logout

página inicial. Para unir a implementação do quiz com a de login, podemos adicionar uma condição na div que engloba o mesmo verificando se o usuário atual existe, tal como na **Listagem 18**.

Listagem 18. Div de quiz com condição para usuário logado.

```
{{#if currentUser}}
  <div class="quiz">
    {{#each quizzes}}
      {{>quiz}}
    {{/each}}
  </div>
{{/if}}
```

Notas de Produção

Uma vez que estamos trabalhando em ambiente de desenvolvimento, todas as coleções de dados são publicadas automaticamente no nosso servidor e assimiladas pelo cliente. Não queremos esse tipo de comportamento quando a aplicação estiver em produção. O mais apropriado seria termos cada usuário subscrevendo somente os dados que ele está trabalhando naquele momento.

Dentro do diretório `.meteor/packages` temos um pacote chamado `autopublish`. Precisamos removê-lo para que essa funcionalidade funcione como queremos. Execute então o comando a seguir de dentro do terminal:

```
meteor remove autopublish
```

Em relação à publicação e deploy do projeto para um ambiente real e na internet, se você não tiver um servidor alocado para isso pode usar o serviço de postagem de aplicações do próprio Meteor. Vá até o terminal e execute o comando:

```
meteor deploy <nome da sua app>.meteor.com
```

Se certifique de substituir o `<nome da sua app>` por um nome real e curto. Ao executá-lo você estará enviando uma solicitação ao Meteor para criar um subdomínio particular, mas para isso ele precisa ser único, logo se o nome que você escolher já existir, uma tela de erro aparecerá informando: "Sorry, that site belongs to a different user" (Desculpe, este site já pertence a um usuário diferente).

Obviamente, este artigo foi apenas um pequeno passo no caminho que ainda há para percorrer com o Meteor. Mas foi o suficiente para fornecer uma impressão completa sobre o potencial desse framework. É importante que você tenha em mente que o Meteor não é uma solução universal e podem existir situações onde a realidade do seu projeto leve a usar outros frameworks. Por exemplo, se o seu sistema não necessita dessa integração entre cliente e servidor automática ou já tem um módulo desenvolvido no servidor e precisa apenas do cliente, então não compensa usar o Meteor, você pode fazer uso de outros frameworks mais robustos para essa finalidade como AngularJS ou Backbone.

O Meteor também tem uma feature muito interessante chamada `Iron Router`, que adiciona roteadores de URLs aos serviços

e páginas configuradas nele, flexibilizando muito a navegação do usuário. Você também pode integrar suas aplicações a web services com restful através da API *RESTivus*, testes unitários através do framework *Velocity* e validação de esquemas por meio da *Collection2*. O segredo é aprender a usar o gerenciador de pacotes *Atmosphere.js* para importar os pacotes e incorporar cada vez mais recursos ao seu sistema.

Autor



Jorge Rodrigues

É especialista em SEO e marketing na web, e freelancer no universo mobile e front-end, tendo domínio sobre tecnologias como CoffeeScript, JavaScript, HTML5 e Meteor.js. Já trabalhou em diversas empresas de TI tendo acumulado mais de dois anos de experiência.



Links:

Página oficial de instalação do Meteor.

<https://www.meteor.com/install>

Página de desenvolvedores do Facebook.

<https://developers.facebook.com/>

Página oficial do MongoDB.

<https://www.mongodb.com/>

FÓRUM DEVMEDIA

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



No novo Fórum da DevMedia você vai encontrar canais específicos de Delphi, ASP.NET, Java, Banco de Dados e Engenharia de Software; além de ter contato com profissionais qualificados da área para troca de informações, sugestões e muito mais.

ACESSE AGORA
www.devmmedia.com.br/forum

Primeiros passos com a Google Dart

Principais tópicos para execução em client e server side

Em meados de 2007 o Google inaugurou a sua primeira linguagem de programação, a linguagem Go ou golang, iniciando um ciclo de lançamentos futuros que desencadeariam em poderosas ferramentas como AngularJS, GWT e App Engine. Inicialmente, as linguagens assumiam características estruturadas e estáticas, em detrimento da ampla adoção que a empresa fazia (e ainda faz) de linguagens como C e C++. Mas com o tempo, a necessidade de abraçar os universos front-end, mobile e server side ao mesmo tempo fez com a empresa criasse uma nova linguagem: a Google Dart.

A Dart é uma linguagem desenhada originalmente para a web, que foi concebida na conferência GOTO na Dinamarca em outubro de 2011, em um projeto fundado pelos desenvolvedores Lars Bark e Kasper Lund. Como toda linguagem client side, a Dart precisou passar por uma bateria de testes junto à ECMA International para verificar seu funcionamento em browsers modernos, tendo assim sua primeira especificação aprovada e liberada para a comunidade.

De acordo com o site do projeto “a Dart foi desenhada para facilmente escrever ferramentas de desenvolvimento para aplicações web modernas e capacitadas para ambientes de alta performance”. Dentre as principais características da linguagem, podemos citar:

- É baseada em compilação de código JavaScript;
- Baseada em classes;
- Orientada a objetos;
- Tem sintaxe baseada na linguagem C;
- Implementa heranças simples;
- E suporta os principais tópicos da OO: interfaces, classes abstratas, genéricos e tipos opcionais.

Neste artigo abordaremos os principais conceitos da nova linguagem, sua sintaxe, principais estruturas programáticas, suas bibliotecas e seu SDK, além da grande quantidade de pacotes públicos disponíveis para impor-

Fique por dentro

A maioria das arquiteturas atuais faz uso de plataformas e linguagens diferentes em ambos os lados cliente e servidor, o que muitas vezes leva a código duplicado e complexidade aumentada, isso sem falar dos custos de manutenção e declínio da produtividade da equipe. Foi pensando nisso que o Google criou a linguagem de programação Dart, com o intuito de facilitar a forma como integramos a aplicação como um todo.

Neste artigo veremos os principais tópicos acerca dessa linguagem: como ela lida com a Orientação a Objetos, sintaxe, responsividade, manipulação de erros, classes, interfaces, dentre outros. No fim, o leitor estará apto a introduzir a linguagem nos seus projetos sem se preocupar em perder o poder que tinha antes ao usar JavaScript e Node.js, uma vez que ela gera a conversão para essas e muitas outras tecnologias.

tação nos seus projetos (conceito semelhante ao de frameworks como Grunt e React).

Configuração do ambiente

Para se trabalhar com a Dart a primeira coisa a se fazer é efetuar o download da linguagem, juntamente com o Dark Editor, a Dart SDK e o web browser Chromium que traz consigo uma VM Dart pré-construída e disponibilizada gratuitamente. Para isso, acesse o site oficial disponível na seção **Links** e clique no botão “Get Started”. Após isso, você será redirecionado para a página de definição da plataforma, conforme mostra a **Figura 1**.

Selecione o seu Sistema Operacional, depois clique na opção “Editor + SDK” para selecionar o instalador completo e no final os itens na seção “Download Dart Editor” irão aparecer para você identificar se o seu SO é 32 ou 64 bits.

Posicione o zip baixado na pasta onde você deseja criar os seus projetos Dart, e descompacte-o lá. Neste arquivo estará contido o executável do Dart Editor, basta executá-lo e você verá uma IDE semelhante à da **Figura 2** aparecer.

Primeiros passos com a Google Dart

Perceba como a interface é muito próxima da IDE Eclipse, isso porque o Google usou o núcleo do projeto Eclipse (que é free e open source) para construir uma nova ferramenta de desenvolvimento. Esse tipo de estratégia é muito usado no mercado para construir IDEs como Spring IDE, MyEclipse e as ferramentas Rational da IBM. O pacote Dart SDK já vem selecionado e

não deve ser removido da IDE para que a linguagem funcione corretamente.

A Dart pode ser usada para criar aplicações que não estejam presas ao HTML, de fora do browser. Para isso, basta integrar o seu ambiente ao Node.js. Porém, como a maioria das aplicações são feitas visando o JavaScript, trataremos de focar num exemplo inicial onde exibimos um menu de

navegação no browser e o usuário poderá clicar no mesmo e ser redirecionado para duas telas: uma para inverter o texto num campo de input e outra com informações sobre o site. Este exemplo nos permitirá entender como a linguagem lida com os eventos de páginas HTML.

A Dart possibilita que criemos várias sortes de aplicações (a maioria com tecnologia Google) como projetos:

- **Console Application:** aplicações de execução direta via terminal de comandos, através da geração de arquivos .bat ou equivalentes de cada SO;
- **App Engine Application:** aplicações focadas na Google App Engine, para execução na nuvem do Google;
- **Chrome App:** aplicações ou plugins para o navegador Google Chrome;
- **Dart Package:** para criar um componente modularizado da Dart;
- **Polymer Web Server:** para hospedar aplicações desenvolvidas com o framework JavaScript Polymer.js;
- **Web Applications:** aplicações web simples, com suporte a JavaScript/bibliotecas, HTML e CSS.

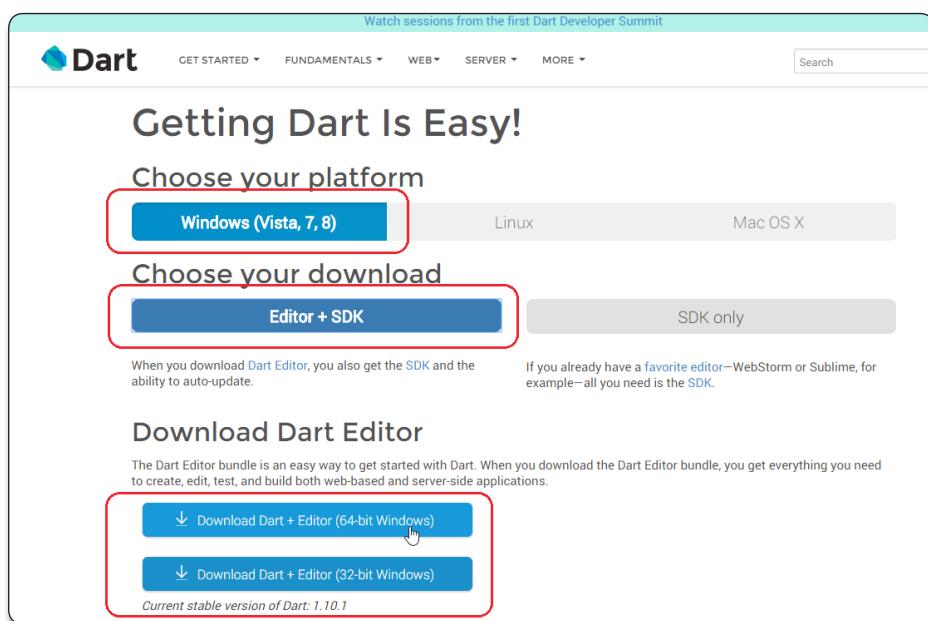


Figura 1. Página de download com opções selecionadas

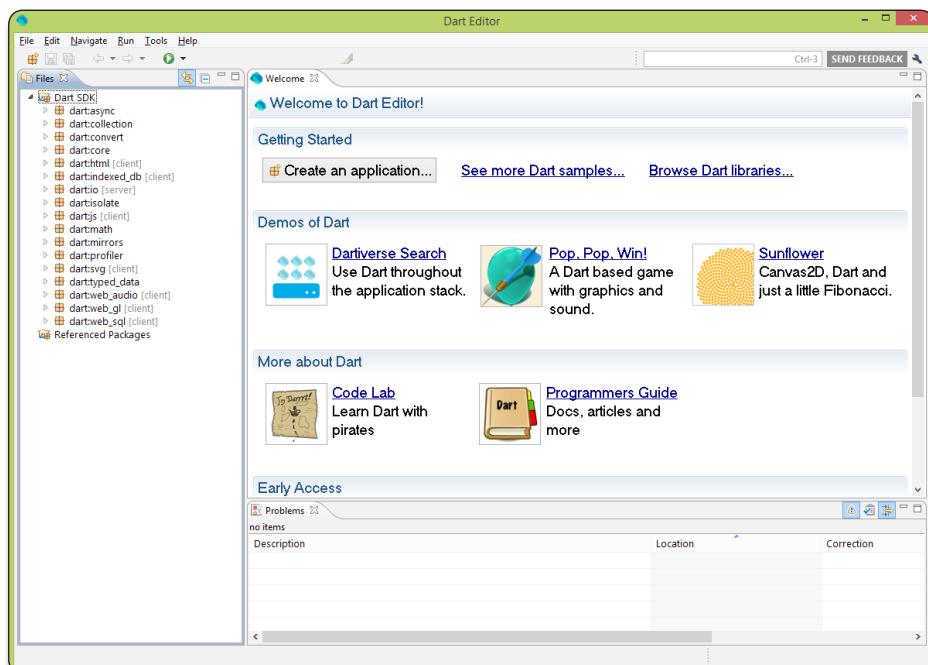


Figura 2. Interface do Dart Editor

Para isso, vá até o menu “File > New Application”. Preencha os campos “Application name” com o nome da aplicação e “Parent directory” com o diretório onde deseja salvá-la. Vamos dar o nome de **AloMundoDart**. Mantenha a checkbox “Generate sample content” marcada para que a wizard de criação já nos traga uma estrutura pré-criada, e selecione a opção “Web application” na lista que aparecer logo abaixo. Clique em “Finish”. Isso irá gerar uma estrutura de projeto igual à da Figura 3.

A pasta lib conterá todos os arquivos de script da Dart, inclusive com dois gerados para o exemplo criado pela wizard. A pasta web guarda todo tipo de conteúdo relacionado ao front-end: imagens, CSS, arquivos HTML, favicons e outros arquivos .dart.

Antes de executar o projeto, façamos algumas alterações. Abra o arquivo `nav_menu.dart` e substitua seu conteúdo pelo representado na **Listagem 1**. Vejamos alguns detalhes da implementação:

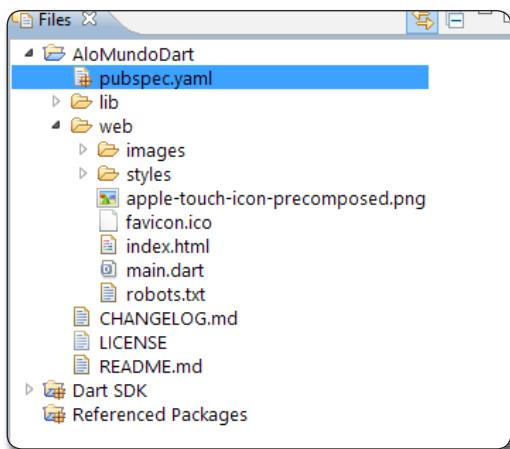


Figura 3. Estrutura de pastas do projeto recém-criado

Listagem 1. Conteúdo do arquivo nav_menu.dart.

```

01 // Copyright (c) 2015, DevMedia. All rights reserved. Use of this source code
02 // is governed by a BSD-style license that can be found in the LICENSE file.
03
04 library nav_menu;
05
06 import 'dart:html';
07
08 initNavMenu() {
09   var navdrawerContainer = querySelector('.navdrawer-container');
10  var appBarElement = querySelector('app-bar');
11  var menuBtn = querySelector('menu');
12  var main = querySelector('main');
13
14 closeMenu(e) {
15   document.body.classes.remove('open');
16   appBarElement.classes.remove('open');
17   navdrawerContainer.classes.remove('open');
18 }
19
20 toggleMenu(e) {
21   document.body.classes.toggle('open');
22   appBarElement.classes.toggle('open');
23   navdrawerContainer.classes.toggle('open');
24   navdrawerContainer.classes.add('opened');
25 }
26
27 main.onClick.listen(closeMenu);
28 menuBtn.onClick.listen(toggleMenu);
29 navdrawerContainer.onClick.listen((event) {
30   if (event.target.nodeName == 'A' || event.target.nodeName == 'LI') {
31     closeMenu(event);
32   }
33 });
34 }
```

- Na linha 4 declaramos o nome da biblioteca que estamos implementando (que também pode ser empacotada e importada como um componente independente em outros projetos). Essa biblioteca será responsável por gerenciar o menu de navegação da nossa página.
- Na linha 6 importamos a biblioteca *html* da Dart. Todos os imports devem ser sempre feitos usando a sintaxe *fabricante:biblioteca*.
- Da linha 9 à linha 12 estamos recuperando os elementos HTML referentes às divs de container, botões e barra de menu.

A função `querySelector()` recebe o seletor do elemento a ser recuperado e varre a página buscando-o, caso exista mais de um, um vetor será criado e associado.

• As funções `closeMenu()` e `toggleMenu()` (linhas 14 e 20) apenas tratam de mudar as classes CSS dos mesmos elementos recuperados para que eles mudem de estilo quando um item de menu for selecionado e desselecionado. A função `remove()`, por sua vez, remove a classe CSS passada por parâmetro do elemento, enquanto `toggle()` testa se a classe já existe, senão ela insere a informada como argumento. Já a função `add()` (linha 24) adiciona a classe informada ao elemento HTML.

• Nas linhas 27 a 29 definimos o ouvinte de click dos componentes de menu para executar as funções de navegação de página. Na Dart, sempre usamos os mesmos atributos dos eventos ouvintes “on” para informar qual evento deve ser executado. Já a função `listen()` da API da Dart recebe uma segunda função como parâmetro para ser executada sempre que o click for efetuado. Veja que a Dart também nos permite criar funções anônimas (linha 30) em vez de defini-las sempre em funções separadas.

Agora abra o arquivo `reverser.dart` que conterá o código de reversão dos textos e adicione o conteúdo da **Listagem 2**.

Listagem 2. Conteúdo do arquivo reverser.dart.

```

01 // Copyright (c) 2015, DevMedia. All rights reserved. Use of this source code
02 // is governed by a BSD-style license that can be found in the LICENSE file.
03
04 library reverser;
05
06 import 'dart:html';
07
08 InputElement get _InputElement => querySelector('#nome');
09 Element get _outputElement => querySelector('#out');
10
11 // Exemplo de gancho no DOM e resposta às mudanças de campos de entrada.
12 initReverser() {
13   // Revele assim que o código inicia.
14   _reverse();
15
16   // Reverte há cada soltura de tecla.
17   _InputElement.onKeyUp.listen(_ >= _reverse());
18 }
19
20 _reverse() {
21   _outputElement.text = _InputElement.value.split('').reversed.join();
22 }
```

Vejamos alguns detalhes:

- Na linha 4 definimos novamente o nome da biblioteca, e na linha 6 a importação da lib *html*.
- Na linha 8 recuperamos o campo de input do nome a ser digitado e adicionamos a um objeto `InputElement` da API da Dart.
- Na linha 9 recuperamos o elemento referente ao campo de output (a Dart disponibiliza essa e outras novas tags HTML) que imprimirá o texto reverso.
- Na linha 20 chamamos a função `_reverse()` que simplesmente acessa o valor do campo de input, quebra-o em um vetor de chars

(split) e chama a propriedade *reversed* que será responsável por reverter as posições do vetor. No fim, chamamos a função *join()* que une o vetor em uma string novamente e o adiciona à propriedade *text* (como se trata de um campo de output, não usamos o “value” dos input, e sim “text”) do elemento.

- Finalmente, na linha 17 implementamos o evento de *onKeyUp* para quando o usuário digita algo no campo e associamos a função *_reverse()* ao mesmo.

Agora precisamos editar o arquivo *main.dart* que representa o escopo inicial de execução do framework, isto é, ele sempre irá carregar os demais arquivos dart a partir deste. Para isso, adicione o conteúdo da **Listagem 3** ao mesmo.

Listagem 3. Conteúdo do arquivo main.dart.

```
01 // Copyright (c) 2015, DevMedia. All rights reserved. Use of this source code
02 // is governed by a BSD-style license that can be found in the LICENSE file.
03
04 import 'dart:html';
05
06 import 'package:AloMundoDart/nav_menu.dart';
07 import 'package:AloMundoDart/reverser.dart';
08 import 'package:route_hierarchical/client.dart';
09
10 void main() {
11   initNavMenu();
12   initReverser();
13
14   // Webapps precisam de routing para ouvir as mudanças na URL.
15   var router = new Router();
16   router.root
17     ..addRoute(name: 'sobre', path: '/sobre', enter: showSobre)
18     ..addRoute(name: 'home', defaultRoute: true, path: '/', enter: showHome);
19   router.listen();
20 }
21
22 void showSobre(RouteEvent e) {
23   // Extremely simple and non-scalable way to show different views.
24   querySelector('#home').style.display = 'none';
25   querySelector('#sobre').style.display = '';
26 }
27
28 void showHome(RouteEvent e) {
29   querySelector('#home').style.display = '';
30   querySelector('#sobre').style.display = 'none';
31 }
```

Vejamos alguns detalhes:

- Nas linhas 6 a 8 vemos a importação dos dois arquivos de script dart que criamos antes, além de um novo “client.dart”. Esse arquivo será herdado da API para implementar o *routing* das páginas, ou seja, lidar com as mudanças de URL de acordo com as ações definidas para cada página.

- Na linha 15 criamos um novo objeto *Router*, que se encarrega de definir as rotas possíveis para a aplicação como um todo. O método *addRoute()* recebe alguns parâmetros, a saber:

- **name:** nome da rota.
- **path:** URL relativa do caminho interno da rota.
- **enter:** recebe a função a ser chamada quando essa rota for acessada.

- **defaultRoute (opcional):** booleano para definir se a rota atual é a padrão.

- Os métodos *showHome* e *showSobre* se encarregam de exibir e esconder as divs de conteúdo da página. Estas devem ser incrementadas à medida que as páginas aumentam na aplicação.

Para finalizar, vamos editar a página HTML *index.html*. Ela é um pouco extensa, então vamos dividir em duas partes: cabeçalho/navegação (**Listagem 4**), e página de conversão/página de sobre.

Listagem 4. Conteúdo do cabeçalho e div de navegação da página index.html.

```
01 <!DOCTYPE html>
02
03 <html>
04 <head>
05   <meta charset="utf-8">
06   <meta http-equiv="X-UA-Compatible" content="IE=edge">
07   <meta name="viewport" content="width=device-width, initial-scale=1.0">
08   <meta name="scaffolded-by" content="https://github.com/google/stagehand">
09   <title>AloMundoDart</title>
10
11   <!-- Add à homescreen pelo Chrome no Android -->
12   <meta name="mobile-web-app-capable" content="yes">
13   <link rel="icon" sizes="192x192"
14     href="images/touch/chrome-touch-icon-192x192.png">
15
16   <!-- Add à homescreen pelo Safari no iOS -->
17   <meta name="apple-mobile-web-app-capable" content="yes">
18   <meta name="apple-mobile-web-app-status-bar-style" content="black">
19   <meta name="apple-mobile-web-app-title" content="Web Starter Kit">
20   <link rel="apple-touch-icon-precomposed"
21     href="apple-touch-icon-precomposed.png">
22
23   <!-- Tile icon for Win8 (144x144 + tile color) -->
24   <meta name="msapplication-TileImage"
25     content="images/touch/ms-touch-icon-144x144-precomposed.png">
26   <meta name="msapplication-Color" content="#3372DF">
27
28   <!-- TODO: Troque as duas referências de folha de estilo para usar o Sass
29   (e tire o transformador sass no arquivo pubspec.yaml. -->
30   <link rel="stylesheet" href="styles/main_gen.css">
31   <!-- link rel="stylesheet" href="styles/main.css" -->
32 </head>
33
34 <body>
35   <header class="app-bar promote-layer">
36     <div class="app-bar-container">
37       <button class="menu">
38     </button>
39     <h1 class="logo">Reversor de Palavras</h1>
40     <section class="app-bar-actions">
41       <!-- Put App Bar Buttons Here -->
42       <!-- e.g <button><i class="icon icon-star"></i></button> -->
43     </section>
44   </header>
45   <nav class="navdrawer-container promote-layer">
46     <h4>Navegação</h4>
47     <ul>
48       <li><a href="/">Home</a></li>
49       <li><a href="/sobre">Sobre</a></li>
50     </ul>
51   </nav>
```

A maior parte dessa listagem comprehende tags de importação de estilo e tags meta, além da estrutura de divs HTML para conter o menu de navegação. Vejamos alguns detalhes:

- As tags meta das linhas 12 e 13 servem para criar um atalho da aplicação nos dispositivos Android. Da mesma forma, as representadas nas linhas 16 a 18 fazem o mesmo para dispositivos iOS. Já na linha 22 temos uma representação deste código para dispositivos que rodam Windows 8/Windows Phone.
- Na linha 25 temos o conteúdo da tag de estilo necessária para incutir Sass nas páginas, caso deseje.
- O restante da implementação traz apenas divs e links para a navegação. A estrutura foi feita baseada no CSS disponibilizado pelo Dart no projeto recém-criado.

A segunda parte da página pode ser encontrada na **Listagem 5** e deve ser adicionada logo após o final da anterior.

Como boa prática de implementação no front-end, estamos importando os arquivos de script da Dart apenas no final, assim ganhamos em performance pois a página já estará carregada quando chegarmos nestes arquivos. Perceba que o atributo type da tag script (comumente preenchido com o valor “text/javascript”) deve ser preenchido com “application/dart” sempre.

Já o src aponta para o arquivo main.dart. A segunda tag de script (linha 33) serve para carregar a API da Dart na página.

Note também que a segunda div deve vir escondida (display:none) para que possamos exibi-la somente quando selecionada no menu. Veja como é fácil incutir navegação na Dart: basta criar divs e escondê-las, deixando que a API se encarrega de redirecionar para onde você mandar.

Listagem 5. Conteúdo das páginas de conversão e sobre.

```
01 <main>
02
03 <div id="home">
04   <p>
05     Nome:<br>
06     <input type="text" id="nome" placeholder="Digite o seu nome...">
07   </p>
08
09   <p>
10     Revertido:<br>
11     <output id="out"></output>
12   </p>
13 </div>
14
15 <div id="sobre" style="display:none">
16   <p>
17     Este é um aplicativo de demonstração HTML básico construído com Dart.
18     Para os desenvolvedores que não querem, ou não podem,
19     usam polymer.dart,
20     este é um ponto de partida opcional.
21   </p>
22 </div>
23 <div class="sass-hidden">
24   <p>
25     Nota: para usar Sass com esta aplicação,
26     mude as referências de folha de estilo na tag &lt;head&gt;;
27     e descomente o transformador sass no arquivo pubspec.yaml.
28   </p>
29 </div>
30 </main>
31
32 <script type="application/dart" src="main.dart"></script>
33 <script data-pub-inline src="packages/browser/dart.js"></script>
34 </body>
35 </html>
```

Conhecimento faz diferença!

The image shows the cover of the magazine 'engenharia de software'. It features several columns of text and small images related to software engineering topics like 'Agilidade', 'Processo', 'SOA', 'Qualidade de Software', and 'Automação de Testes'. A large red starburst graphic on the right side contains the text '+ de 290 vídeos para assinantes'.

Faça já sua assinatura digital! | www.devmedia.com.br/es

Faça um upgrade em sua carreira

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



DEVMEDIA

Primeiros passos com a Google Dart

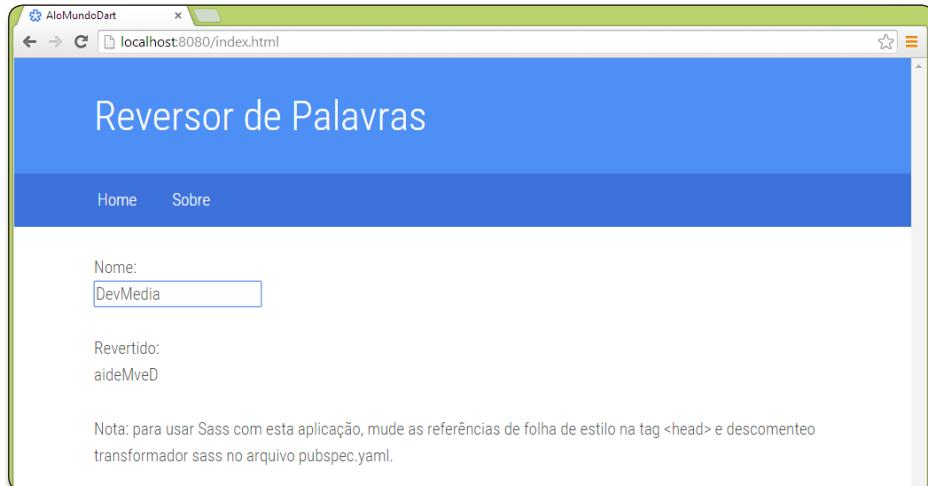


Figura 4. Resultado da página de reversão



Figura 5. Navegando à página de sobre

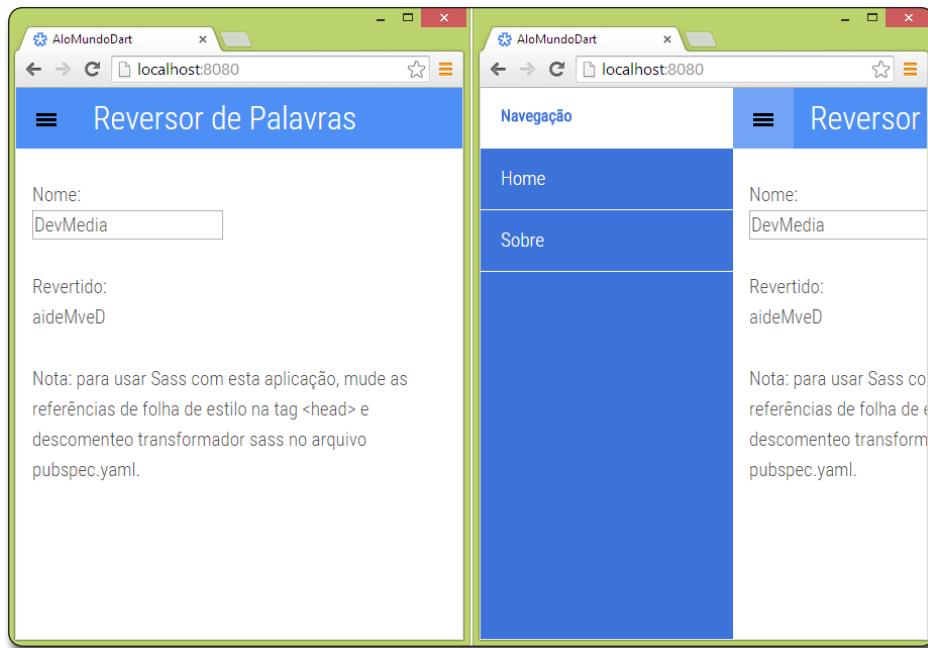


Figura 6. Telas da aplicação em dispositivo menor

Após isso, salve todos os arquivos e clique no botão *Run* no topo da IDE ou selecione o atalho Ctrl + R. O resultado deverá ser igual ao das **Figuras 4 e 5**.

Note que o Editor abrirá uma janela interna do Google Chrome. Você não precisa ter o browser instalado, já que ele traz uma instância interna do mesmo no pacote de instalação. Essa abordagem é usada pelo Google para evitar fazer testes em browsers internos às IDEs (como no Eclipse, por exemplo), possibilitando testes mais reais e automatizados. A comunicação com frameworks de teste como o Selenium é fácil de ser feita nessa versão do Chromium.

Veja que na barra de URLs do browser interno ele abriu a aplicação no endereço <http://localhost:8080>. Essa é a URL padrão do framework, pois ele faz uso do Node.js para compilar e executar o JavaScript no servidor. Por acaso se não estiver conseguindo executar ou aparecer algum erro de porta em uso, verifique se não está executando alguma ferramenta que usa a porta 8080, como os servidores Java Tomcat e JBoss, por exemplo.

Ainda sobre a URL, veja que na tela *Sobre* temos a URL com a respectiva terminação. Isso acontece porque definimos que a ação é exibir a div com a propriedade *name* de valor *sobre*.

A Dart também gera conteúdo responsivo automaticamente. Logo, se executarmos a mesma tela em um dispositivo smartphone ou tablet, teremos como resultado a tela exibida na **Figura 6**.

Principais características

A Dart inaugurou uma lista de novos conceitos no mundo da programação. Vejamos alguns deles.

Modularidade e Namespace

O conceito de modularidade é baseado na divisão em componentes menores e independentes. Além disso, cada módulo precisa ter uma responsabilidade única e fornecer estruturas que permitam se conectar a ele. Por exemplo, se tivermos um sistema hospitalar e desejarmos dividi-lo em módulos que possam ser acoplados a outros sistemas de terceiros, teríamos um

módulo para o almoxarifado, outro para o controle de consultas, mais um para o cadastro de exames, e por aí vai.

Esse tipo de recurso é muito usado por softwares de clientes que precisam de alguns componentes enquanto outros do mesmo negócio não. No universo de programação, essa divisão leva o nome de “decomposição”. Para que cada parte do sistema seja decomposta ela precisa passar por um ciclo de ações: criação, mudança, teste, uso e substituição separada.

A modularidade na Dart é feita através dos pacotes, bibliotecas e classes:

- Uma **biblioteca** expõe as funcionalidades como um conjunto de interfaces e esconde a implementação do resto do mundo. Como um conceito, é muito similar à separação de responsabilidades que temos na Programação Orientada a Objetos, por exemplo. Como resultado, o usuário passa a ter menos código concentrado e mais facilidade na hora de manter o mesmo, já que as mudanças se concentram agora sempre num componente só. Além disso, o conceito de biblioteca passa a não mais englobar somente APIs server side, mas também frameworks front-end.
- Um **pacote** é um simples diretório que contém um arquivo chamado pubspec.yaml e inclui em si uma quantidade x de bibliotecas e recursos. Este arquivo contém informações importantes sobre o pacote como seus autores, além de suas dependências em relação a outros pacotes. Vejamos na **Listagem 6** o conteúdo deste arquivo criado no nosso exemplo de AloMundo. Perceba que ele funciona como uma espécie de Google Doc, porém especificamente para a Dart; além disso, estes são apenas campos básicos, é possível encontrar uma lista das demais opções no site oficial da linguagem (vide seção **Links**). Esse arquivo é obrigatório para que o seu Sistema Operacional reconheça o tipo de tecnologia que irá operar ali.

Antes de ser usado, um pacote deve sempre ser publicado em um “sistema de gerenciamento de pacotes”, que está disponível como um recurso online chamado *pub* (veja na seção **Links** a URL do pub oficial da Dart). Em relação à recuperação dos mesmos pacotes no pub, uma aplicação utilitária de mesmo nome pode ser usada para tal. Esta usa informações sobre as dependências do pubspec.yaml para recuperar todos os pacotes necessários dos seguintes locais:

- Pacotes recentemente atualizados no site da pub Dart;
 - Repositório Git;
 - Diretório no filesystem local.
- As **classes** podem ser representadas usando funções comuns ou protótipos de herança.

Já os **namespaces** são containers para todos os membros de uma biblioteca. Um namespace é definido pelo nome da biblioteca. Ou seja, se uma biblioteca é implicitamente nomeada então ela tem um namespace vazio. Isso resulta muitas vezes em conflitos ao tentar importar bibliotecas com os mesmos namespaces. Tais conflitos podem ser evitados com uma cláusula prefixada (*as*) e um nome de prefixo.

Veja a **Listagem 7**. Nela, temos um exemplo de biblioteca implicitamente nomeada na qual todos os recursos da dart:html estão disponíveis através do escopo da nossa biblioteca com o prefixo dom.

Listagem 6. Estrutura de diretórios padrão do projeto.

```
01 name:'AloMundoDart'  
02 version: 0.0.1  
03 description: >  
04 A mobile-friendly web app with routing, responsive CSS, and (optional) Sass  
05 support.  
06 #author: <your name> <email@example.com>  
07 #homepage: https://www.example.com  
08 environment:  
09 sdk:>=1.0.0 <2.0.0'  
10 dependencies:  
11 browser:>=0.10.0 <0.11.0'  
12 sass:>=0.4.0 <0.5.0'  
13 script_inliner:>=1.0.0 <2.0.0'  
14 route_hierarchical:>=0.5.0 <0.7.0'  
15 transformers:  
16 - script_inliner  
17 # sass:  
18 # style: compressed
```

Listagem 2. Exemplo de biblioteca implicitamente nomeada.

```
01 /**  
02 * Biblioteca nomeada implicitamente.  
03 * A lib dart:core é automaticamente importada.  
04 */  
05 import 'dart:html' as dom;  
06 /**  
07 * Recupera [Elemento] por [id].  
08 */  
09 dom.Element getById(String id) => dom.querySelector('#$id');
```

Obviamente, todas as marcações e seletores só serão reconhecidos em um ambiente Dart. O browser não consegue interpretar essa notação. Veja que na linha 5 fazemos uso da cláusula “as” para importar o recurso (biblioteca HTML da Dart) de forma dinâmica. Na linha 9 fazemos a recuperação do elemento id pelo seu atributo id da HTML. Isso é bem semelhante à forma como o jQuery faz a seleção de seus atributos, por exemplo. Contudo, quando for compilado para JavaScript, todo esse código perde as informações das bibliotecas e temos um arquivo bem diferente sendo gerado.

A Dart implementa o conceito de encapsulamento através da privacidade. Cada membro ou identificador de uma biblioteca tem um dos dois níveis de acesso: público ou privado. Membros privados são visíveis apenas de dentro da biblioteca na qual eles são declarados. Por outro lado, os membros com um acesso público são visíveis em todos os lugares. A diferença entre eles é o prefixo sublinhado (_), como mostrado no código da **Listagem 8**.

O código mostra uma biblioteca de animação com duas classes e uma variável. A classe de Animação e a variável animationSpeed são públicas e, portanto, visíveis de fora da biblioteca. A classe _AnimationLibrary é privada e só pode ser utilizada na biblioteca.

Listagem 8. Exemplo de encapsulamento na Dart.

```
01 // Biblioteca Animation.  
02 library animation;  
03 // Classe publicamente disponível em qualquer lugar.  
04 class Animation {  
05 // ...  
06 }  
07 // Classe visível apenas dentro da biblioteca.  
08 class _AnimationLibrary {  
09 // ...  
10 }  
11 // Variável publicamente disponível em qualquer lugar.  
12 var animationSpeed;
```

Acessos públicos podem ser controlados com as extensões *show* e *hide* na declaração de importação. Use a seguinte extensão *show* com uma classe específica, que estará disponível dentro da biblioteca em que é importada:

```
import 'animation.dart' as animation show Animation;  
main() { new animation.Animation(); }
```

O prefixo **animation** na declaração de importação define o namespace para importar a biblioteca *animation.dart*. Todos os membros da biblioteca *animation.dart* estão disponíveis no namespace global através deste prefixo.

Já a extensão *hide* especificada na classe fará com que a mesma esteja inacessível de dentro da biblioteca na qual foi importada. Isso é interessante quando precisamos que uma classe só seja acessa de dentro da própria API, como classes de segurança, conexão com banco, etc.:

```
import 'animation.dart' as animation hide Animation;  
main() { var speed = animation.animationSpeed; }
```

Funções e Closures

Na Dart é possível criar **funções** via variável, isto é, criamos uma referência a uma função e a atribuímos para uma variável, conforme pode ser observado na **Listagem 9**.

Listagem 9. Exemplo de função com associação de variáveis a funções.

```
01 library function_var;  
02 // Retorna a soma de [a] e [b]  
03 soma(a, b) {  
04   return a + b;  
05 }  
06 // Operação  
07 var operacao;  
08 void main() {  
09   // Atribui a referência à função [soma]  
10   operacao = soma;  
11   // Execute operacao  
12   var resultado = operacao(2, 1);  
13   print("Resultado será ${resultado}");  
14 };  
15 }
```

O resultado será 3. Veja que criamos uma função simples de soma de dois valores e adicionamos a função a uma variável (*operacao*, linha 10) chamando-a em vez de à função *soma()*. Na linha 13 imprimimos o resultado da soma. Veja que as impressões no console JavaScript são feitas via método *print()* e os valores a se concatenar via operador *{}\$*.

A Dart ainda lida com a passagem de funções como argumentos para outras funções; retorno de funções como resultado de outras (*return*) e consegue salvar funções em estruturas de dados, como vetores, por exemplo.

A função pode ser criada em âmbito global ou no âmbito de uma outra função. Uma função que pode ser referenciada com um acesso para as variáveis no seu escopo léxico é chamada de **closure**. Vejamos o código da **Listagem 10**.

Listagem 10. Exemplo de closure na Dart.

```
01 library function_closure;  
02  
03 // Função retorna uma closure.  
04 calcular(base) {  
05   // Contador  
06   var cont = 1;  
07   // Função interna - closure  
08   return () => print("Valor será ${base + cont++}");  
09 }  
10 void main() {  
11   // A função externa retorna a interna  
12   var f = calcular(2);  
13   // Agora chamamos a closure  
14   f();  
15   f();  
16 }
```

A estrutura para criar uma closure é a mesma para as funções comuns. A diferença é que a closure é criada e retornada de dentro de uma função já existente. Ela funciona como uma função anônima que pode ser criada facilmente sem a necessidade de todo o aparato de palavras-chave para formar as funções comuns. Veja que na linha 8, após criar a variável contadora, estamos associando o ato de imprimir a soma e contador a uma função vazia *()*. O operador *=>* serve para fazer essa associação, muito comum em estruturas como lambdas. Após isso, sempre que quisermos chamar essa closure basta retornar para uma variável e efetuar as chamadas: *f()* (linhas 14 e 15).

Neste exemplo, os resultados serão 3 e 4, respectivamente.

Classes

No mundo real, encontramos muitos objetos individuais, todos do mesmo tipo. Há muitos carros com a mesma marca e modelo. Cada carro foi construído a partir do mesmo conjunto de projetos. Todos eles contêm os mesmos componentes e cada um é uma instância da classe de objetos conhecida como *Carro*.

O conceito de classe na orientação a objetos é o mesmo na Dart. Atributos, métodos, classes internas e construtores se mantêm na

linguagem. Vejamos a **Listagem 11**, onde temos a representação de um objeto do tipo Carro.

Listagem 11. Representação da classe Carro na Dart.

```
01 library carro;
02 //Classe abstrata [Carro] não pode ser instanciada.
03 abstract class Carro {
04   String cor;
05   double velocidade;
06   double capacidade;
07
08   // Construtor da classe
09   Car(this.cor, this.capacidade);
10  // Move o carro de acordo com a [velocidade]
11  void move(double velocidade) {
12    this.velocidade = velocidade;
13  }
14  // Para o carro.
15  void stop() {
16    velocidade = 0.0;
17  }
18}
```

Vejamos alguns detalhes da listagem:

- Na linha 3 declaramos a classe sendo de um tipo abstrato (`abstract`). Isso significa que essa classe não poderá nunca ser instanciada (`new`), e sim somente servir como classe pai de outras.
- Nas linhas 4 a 6 declaramos os atributos, todos globais e públicos (não é preciso usar palavras reservadas para isso). O tipo `String` pertence à API da Dart e funciona da mesma forma que em outras linguagens OO (Java, C#, etc.). Os tipos primitivos (`int`, `double`, `char`, `boolean`...) também se mantêm.
- Na linha 9 criamos o construtor da classe que recebe os dois parâmetros para associar aos atributos de cor e capacidade. Veja que não precisamos fazer a associação dos valores manualmente, a própria Dart faz isso para gente automaticamente, simplificando muito o código.
- Os métodos também são públicos. A palavra `void` define que eles não têm nenhum retorno, apenas executando o código interno.

Mixins

A Dart tem um sistema de herança baseado em mixins, que permitem que o corpo de certas classes possa ser reusado em classes hierárquicas. Vejamos o código da **Listagem 12**.

A classe `Reboque` é independente da classe `Carro`, mas pode aumentar a capacidade de carregar peso do carro. Para adicionar um mixin de `Reboque` à classe `CarroPassageiro` podemos usar a palavra-chave `with` seguida pela classe mixin `Reboque`, como no código da **Listagem 13**.

Com a adição da cláusula `with` na linha 5 temos agora assimilada a funcionalidade de adicionar mais peso ao reboque do carro (linha 15). Para isso, não precisamos mudar nada na classe `CarroPassageiro`, basta adicionar o mixin e os métodos serão automaticamente carregados sem a necessidade de uma herança explícita (`extends`).

Listagem 12. Exemplo de classe a ser mixada.

```
01 library reboque;
02 // O reboque
03 class Reboque {
04   // Acessa a informação de [carga] do carro
05   double carga = 0.0;
06   // O reboque pode carregar tanto de [peso]
07   void carregar(double peso) {
08     // O carregamento aumenta com pesos extra.
09     carga += peso;
10   }
11 }
```

Listagem 13. Exemplo de mixin na Dart.

```
01 library passenger_carro;
02 import 'carro.dart';
03 import 'reboque.dart';
04 // Carro de passageiro com reboque.
05 class CarroPassageiro extends Carro with Reboque {
06   // Nº max de passageiros.
07   int maxPassageiros = 4;
08   /**
09    * Cria um [Passageiro] com a [cor], [carga] e [maxPassageiros].
10   * Podemos usar o [Reboque] para carregar [pesoExtra].
11   */
12   Passageiro(String cor, double carga, this.maxPassageiros,
13   {double pesoExtra:0.0}) : super(cor, carga) {
14     // Só podemos carregar peso extra com [Reboque]
15     carregar(pesoExtra);
16   }
17 }
```

Os mixins são semelhantes às interfaces e só podem ser definidos via classe, tendo algumas restrições no seu uso, a saber:

- Não podem ter construtores definidos;
- A superclasse de um mixin só pode ser um `Object`;
- Eles não podem ter chamadas ao operador `super`.

Coleções e Genéricos

A Dart suporta o uso de vetores na forma de classes de `List`. Digamos que precisamos usar uma lista para salvar informações temporárias. O dado que você coloca na lista depende do contexto do código. A lista deve conter diferentes tipos de dados ao mesmo tempo, conforme demonstrado pela **Listagem 14**.

Listagem 14. Exemplo de lista na Dart.

```
01 // Lista de dados aleatórios
02 List lista = [1, "Letra", {'test': 'errado'}];
03
04 // Item qualquer
05 double item = 1.53;
06
07 void main() {
08   // Add o item ao array
09   lista.add(item);
10   print(lista);
11 }
```

Neste exemplo adicionamos dados de diferentes tipos ao mesmo array. Quando o código for executado vermos o resultado [1, Letra, {test: errado}, 1.53] ser impresso no console. Esse tipo de funcionalidade é permitido em várias linguagens de programação. Entretanto, não é interessante ter dados conflitando em uma estrutura que pode dar erros mais à frente. Para evitar esse tipo de comportamento podemos verificar qual tipo de dado está vindo na lista através do operador `is` (Listagem 15).

Listagem 15. Exemplo de lista checada na Dart.

```
01 // Array de dados String
02 List parts = ['roda', 'ignição', 'motor'];
03
04 // Item qualquer
05 double item = 1.53;
06
07 void main() {
08   if (item is String) {
09     // Add o item ao array
10     parts.add(item);
11   }
12   print(parts);
13 }
```

Agora o resultado impresso será `[roda, ignião, motor]`. Dessa forma, garantimos que nenhum tipo de dado que não seja String seja adicionado ao nosso array. Agora, imagine como seria se tivéssemos centenas de arrays no nosso projeto, ao ter de fazer esse tipo de verificação sempre que quisermos criar ou manipular um. É muito trabalho desnecessário. Para resolver isso podemos usar um analisador estático de código que faz o trabalho todo. Chamamos essas estruturas de Genéricos. Vejamos na Listagem 16 como ficaria nosso código.

Dessa forma, caso o programador tente informar algum valor que não seja String no vetor, receberá um erro informando a tipagem errada.

Erros e Exceções

Existe sempre uma confusão muito grande entre a diferença do que é um erro e uma exceção nas linguagens de programação. Erros geralmente acontecem precedidos de falhas graves no fluxo dos algoritmos que impedem a continuação do sistema, como por exemplo o estouro de memória ou quando uma VM não consegue encontrar o recurso (classes, arquivos, etc.) que foi solicitado. Já as exceções são manipuláveis via código, o programador pode capturá-las, entender o que aconteceu e então tomar uma decisão programática.

Vejamos o código da Listagem 17. Ele representa um exemplo onde um eventual erro acontece na Dart.

Listagem 16. Exemplo de lista com Genéricos na Dart.

```
01 // Array de dados String
02 List<String> parts = ['roda', 'ignião', 'motor'];
03
04 // Ordinary item
05 double item = 1.53;
06
07 void main() {
08   // Add o item ao array
09   parts.add(item);
10   print(parts);
11 }
```

Listagem 17. Exemplo de código com "erro" na Dart.

```
01 main() {
02   // Lista de tamanho fixo
03   List lista = new List(5);
04   // Preenche a lista com os valores
05   for (int i = 0; i < 10; i++) {
06     lista[i] = i;
07   }
08   print("Resultado será ${lista}");
09 }
```

Estamos criando uma lista simples com a classe `List` de um tamanho fixo e preenchendo com os valores do inteiro contador em um loop que contém mais itens do que o tamanho máximo permite. Ao executar este código teremos uma exceção semelhante à da Figura 7.

Este erro ocorreu porque foi realizada uma condição de violação em nosso código quando tentamos inserir um valor na lista em um índice fora do intervalo válido. Principalmente, esses tipos de falhas ocorrem quando o contrato entre o código e a API chamada foi quebrado. No nosso caso, `RangeError` indica que o pré-requisito foi violado. Há um monte de erros no Dart SDK como `CastError` (Erro de conversão de um tipo para outro inválido), `NoSuchMethodError` (Quando não é possível encontrar um método informado), `UnsupportedError` (Quando um tipo de dado não é suportado), `OutOfMemoryError` (Quando não há memória suficiente para carregar objetos) e `StackOverflowError` (Quando a pilha de memória estoura o limite máximo). Além disso, existem muitos outros erros que você pode encontrar no arquivo `errors.dart` como uma parte da biblioteca `dart.core`. Todas as classes de erro herdam da classe `Error` e podem retornar informações de rastreamento de pilha para ajudar a encontrar a causa dos mesmos rapidamente. No exemplo anterior, o erro aconteceu na linha 6 do principal método no arquivo `.dart`.

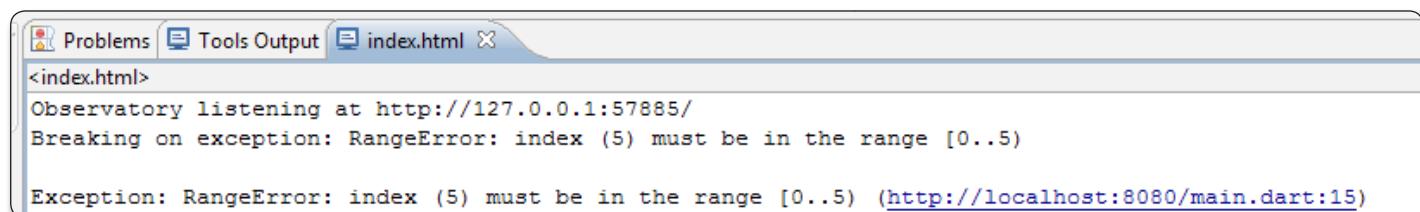


Figura 7. Tela de erro sendo impresso no console da Dart

Os erros podem ser detectados via código, mas são difíceis de manipular, dada a sua natureza imprevista. O mais ideal é sempre conhecer bem a API de erros da Dart para evitar as situações, que são bem exclusivas.

Exceções, ao contrário de erros, são feitas para serem capturadas e geralmente carregam informações sobre a falha, mas não incluem as informações de rastreamento de pilha. Exceções acontecem em situações recuperáveis e não param a execução de um programa. Você pode lançar (`throw`) qualquer objeto não nulo como uma exceção, mas é uma boa prática criar uma nova classe de exceção que implemente a classe abstrata `Exception` e sobrescrever o método `toString` da classe `Object`, a fim de fornecer informações adicionais. Uma exceção deve ser tratada dentro das cláusulas `catch` ou propagadas para as chamadas exteriores. Vejamos na **Listagem 18** um exemplo de código sem o uso de exceções, que faz acesso a um arquivo, uma vez que esse tipo de operação é uma das mais propensas a erro nas linguagens de programação em detrimento do acesso a diretórios, permissões, tamanhos dos arquivos, etc.

Listagem 18. Exemplo de código com exceção sem tratamento.

```
01 import 'dart:io';
02 main() {
03   // URI do arquivo
04   Uri uri = new Uri.file("teste.json");
05   // Checa a uri
06   if (uri != null) {
07     // Cria o arquivo
08     File file = new File.fromUri(uri);
09     // Checa se o arquivo existe
10     if (file.existsSync()) {
11       // Abra o arquivo
12       RandomAccessFile random = file.openSync();
13       // Checa se o arquivo foi aberto
14       if (random != null) {
15         // Lê o arquivo
16         List<int> conteudoNaoPronto = random.readSync(random.lengthSync());
17         // Checa o conteúdo não pronto
18         if (conteudoNaoPronto != null) {
19           // Converte para String
20           String conteudo = new String.fromCharCodes(conteudoNaoPronto);
21           // Imprime o resultado
22           print('Conteúdo: $conteudo');
23         }
24         // Fecha o arquivo
25         random.closeSync();
26       }
27     } else {
28       print ("Arquivo não existe");
29   }
30 }
31 }
```

O resultado da execução será `conteúdo: [name: teste, length: 150]`. Vejamos alguns detalhes do código:

- Na linha 1 importamos a biblioteca IO da Dart, que se encarrega de todas as operações de Entrada/Saída, incluindo acesso a arquivos e serialização.
- Na linha 4 carregamos a URI (URL interna) do arquivo. O leitor pode usar qualquer arquivo que desejar para efetuar o teste.

- O método `fromUri()` da linha 8 se encarrega de carregar o arquivo em memória para ser manipulado.
- O método `existsSync()` da linha 10 verifica se o arquivo de fato existe. Esse é o primeiro tratamento de erro que fazemos, isto é, em vez de manipularmos uma exceção diretamente, fazemos um teste para cada arquivo vendo se ele existe antes de usá-lo.
- Na linha 12 abrimos o arquivo para pegar seus bytes e na linha 16 lemos o seu conteúdo a fim de iterar sobre os dados do mesmo.
- Na linha 22 imprimimos o conteúdo que foi concatenado, fechando o arquivo na linha 25.

É interessante observar que, mesmo que o conteúdo seja impresso com sucesso e nenhuma exceção aconteça pelas precauções que tomamos, ainda assim na linha 25, ao fechar o arquivo, corremos o risco de receber uma exceção por várias razões: o arquivo foi removido por outrem, falha na conexão de rede, etc.

Em vista disso, para se ter uma maior segurança nesse fluxo e manter a organização do código, vamos evoluir o mesmo para o que está contido na **Listagem 19**.

Listagem 19. Exemplo de código com exceção e tratamento.

```
01 import 'dart:io';
02 main() {
03   RandomAccessFile random;
04   try {
05     Uri uri = new Uri.file("teste.json");
06     File file = new File.fromUri(uri);
07     random = file.openSync();
08
09     List<int> notReadyContent = random.readSync(random.lengthSync());
10
11     String conteúdo = new String.fromCharCodes(notReadyContent);
12
13     print('Conteúdo: $conteúdo');
14   } on ArgumentError catch(ex) {
15     print('Erro de argumento inválido');
16   } on UnsupportedError catch(ex) {
17     print('URI não pode referenciar um arquivo');
18   } on FileSystemException catch(ex) {
19     print('Arquivo não existe ou está inacessível');
20   } finally {
21     try {
22       random.closeSync();
23     } on FileSystemException catch(ex) {
24       print('Arquivo não pode ser fechado');
25     }
26   }
27 }
28 }
```

As mudanças não foram tão grandes, mas fazem uma grande diferença na execução final. O bloco `try/on/catch` é a estrutura usada para capturar as exceções e a sintaxe é bem semelhante à das outras linguagens. O nome do erro/exceção que você imagina que possa acontecer no código deve sempre vir declarado após a cláusula `on`, evitando sempre usar superclasses genéricas para capturar as exceções específicas. Veja que no bloco `finally` (atrelado

ao *catch*, que será obrigatoriamente executado ao final da sentença) efetuamos o fechamento do arquivo, mas caso algum erro aconteça nessa parte também faremos a validação e tratamento do erro. Dessa forma, garantimos um código organizado e livre de falhas que impeçam o sistema de continuar executando.

Existem muitos outros recursos que a linguagem Dart disponibiliza, tais como Anotações (para diminuir a quantidade de código em XML), Proxies dinâmicos (para aumentar a performance salvando dados em cache), reflection (para modificar as próprias estruturas programadas em tempo de execução), bem como outros conceitos da Orientação a Objetos (polimorfismo, interfaces, composição, etc.).

Contudo, com o conteúdo aqui exposto o leitor estará apto a criar aplicações mais robustas usando a Dart, seu SDK e o poderoso Editor que a mesma disponibiliza. Para publicar suas aplicações, você pode utilizar a página do Google Code ou do próprio GitHub, bem como participar do projeto open source e ajudar a melhorar cada vez mais a linguagem. A Dart também se integra facilmente a bancos de dados, web services, outros frameworks JavaScript, como jQuery, Prototype.js e AngularJS. Agora é com você e a sua criatividade. 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:

Página de download oficial da Dart.

<https://www.dartlang.org/downloads>

Página do repositório de pacotes da Dart.

<https://pub.dartlang.org/>

CURSOS ONLINE

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

Feita para Desenvolvedores de Software e DBAs



CONHEÇA OS CURSOS MAIS RECENTES:

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

Para mais informações :

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

(21) 3382-5038



DEV MEDIA

Testes automatizados com o framework Selenium

Aprenda a automatizar os testes das suas telas front-end e ganhe em produtividade e escalabilidade

A automação de testes existe para, através de ferramentas e estratégias, reduzir ao máximo o envolvimento humano em atividades manuais repetitivas, como o cadastro de um cliente ou o login/logout em uma aplicação.

O principal objetivo de um teste automatizado é ser usado futuramente em testes de regressão. Esse termo, por sua vez, é usado para quando precisamos efetuar ciclos de re-teste de uma ou várias funcionalidades, com a intenção de identificar problemas ou defeitos introduzidos pela adição de novas funcionalidades ou pela correção de alguma já existente.

Levando em consideração um exemplo real, quando um projeto de software não utiliza testes automatizados em sua concepção, sempre que uma alteração acontece em determinada tela ou módulo, um novo ciclo de testes manuais precisa ser refeito e, assim, sucessivamente até que o projeto esteja terminado. O problema aumenta mais ainda quando o projeto não tem fim e exige manutenção constante, implicando consequentemente em mais custos com uma equipe de teste e possivelmente em entregas sem qualidade, já que não houve tempo o suficiente para testar e re-testar tudo.

Em contrapartida, a utilização de testes automatizados acelera os ciclos do software, uma vez que o teste passa a ser executado por outro software inteligente, que acessa as telas e reexecuta os mesmos testes nos dando relatórios rápidos do que deixou de funcionar. Além disso, estamos menos suscetíveis a erros, já que a máquina nunca erra, ao contrário de termos humanos testando.

Existem várias abordagens diferentes para a automação de testes. Os tipos são normalmente agrupados de

Fique por dentro

A automação de testes soma praticamente 20% do tempo de trabalho de um testador (ou desenvolvedor testador) e economiza várias horas de manutenção e de testes de regressão para verificar se tudo está funcionando há cada nova release de software lançada. O Selenium se popularizou como um dos principais frameworks de automação de testes do mercado, principalmente pelo seu suporte a linguagens como Python, Ruby, Java, C# e Node.js.

Neste artigo aprenderemos a configurar o Selenium de duas formas: via IDE Eclipse e via Selenium IDE. Você verá como gravar testes nos browsers, salvar scripts, re-executá-los, além de comunicar seus testes com JavaScript, jQuery e HTML5. Ao final deste, o leitor estará apto a executar testes síncrona e assíncronamente via API, linha de comando ou interface gráfica, além de entender que tipo de teste se encaixa melhor em cada realidade de software.

acordo com a maneira com a qual os testes se integram e interagem com o software. Um testador pode construir seu teste baseando-se em dois conceitos:

1. Interface gráfica

a. **Gravação/Execução (Capture/Playback):** os testes são feitos diretamente na interface gráfica da aplicação: HTML, XML, Swing, etc. As ferramentas geralmente disponibilizam recursos para efetuar a gravação (capture) das ações feitas nas telas que serão convertidas para uma linguagem de script inteligível pela mesma, a qual poderá futuramente ser reexecutada (playback).

b. **Dirigido a dados (Data-Drive):** representa uma abordagem mais dirigida aos dados no sentido de gravar as ações do

usuário, porém sempre fornecendo dados distintos para deixar o teste mais próximo da realidade.

c. **Dirigido a palavras-chave (Keyword-Driven)**: são testes com alto nível de abstração, onde até mesmo usuários comuns podem criar instruções para iniciar um teste, uma vez que o mesmo será baseado em palavras-chave. Cada palavra-chave representa um comando da ferramenta.

2. Lógica de negócio

a. **Linha de comando (Command Line Interface)**: fornece uma interface com um mecanismo de interação com a aplicação por meio de um prompt cmd do SO e os comandos podem ser enviados de lá.

b. **API (Application Programming Interface)**: fornece bibliotecas ou um conjunto de classes para permitir que outras aplicações acessem a interface do teste sem necessariamente ter de conhecer como eles foram feitos.

c. **Test Harness (Equipamentos de teste)**: é um teste específico que visa o teste único e exclusivo da lógica de negócio. Não é permitida interação alguma com as interfaces gráficas.

O sucesso desse tipo de teste depende de identificar e alocar elementos da GUI (*Graphical User Interface*) na aplicação a ser testada e então executar operações e verificações nestes mesmos elementos para que o fluxo do teste seja completado com sucesso.

O Selenium é um framework de testes automatizados portável para aplicações web desenvolvido em 2004 por Jason Huggins como um projeto interno da empresa ThoughtWorks. Ele fornece todo o aparato para capturar e reproduzir os testes via scripts sem a necessidade de aprender nenhuma linguagem de scripting. Também disponibiliza uma linguagem de teste específica para linguagens populares como Java, C#, Groovy, Perl, PHP, Python e Ruby. Os testes podem ser executados em browsers web modernos e os deploy de aplicação rodam tanto em Windows, quanto em Linux e Macintosh.

Dentre a lista de funcionalidades fornecida pelo framework, temos:

- Preenchimento de caixas de texto;
- Checagem de checkboxes e botões de rádio;
- Click em botões e links;
- Navegação de e para <iframe>'s;
- Navegação de e para novas janelas/tabs criadas por links;
- Intereração com a maioria dos elementos como um humano faz.

Na lista de funcionalidades que ele **não** suporta, temos:

- Intereração com Flash, PDF ou Applets;
- Intereração com quaisquer objetos colocados dentro das tags HTML <object> e <embed>.

Neste artigo exploraremos os principais recursos desse framework, através da construção de exemplos reais usando as linguagens JavaScript, jQuery e Java em conjunto com HTML5 e testes efetuados no Google Chrome e Firefox. Veremos desde a configuração e instalação da ferramenta, bem como as principais diferenças entre usar o Selenium embarcado (com uma linguagem de programação) e o Selenium IDE (diretamente no browser Firefox como Add-On).

Configuração do ambiente

O primeiro requisito obrigatório para executar o Selenium no seu Sistema Operacional é ter uma versão recente do JDK (*Java Development Kit*) instalada. Para verificar se você já tem alguma, basta

acessar o terminal de comandos e digitar o seguinte comando:

```
java -version
```

Se você vir um número com a versão do Java sendo impresso, então você já o tem instalado. Caso contrário, acesse a página de downloads do mesmo (seção **Links**) e efetue os passos para instalação (você encontrará tutoriais de como instalar no próprio site da Oracle caso tenha alguma dúvida).

Em seguida, precisamos efetuar o download do Eclipse (seção **Links**), que é a IDE mais utilizada para desenvolver com o Selenium. Selecione sempre a opção “*Eclipse IDE for Java EE Developers*”, dessa forma teremos uma versão da IDE que também desenvolve para a web. Certifique-se também de baixar a versão correspondente com o seu Sistema Operacional (32/64 bits). Quando finalizar o download, move o arquivo zip baixado e o descompacte em um local de sua preferência. Como não é preciso instalá-lo, basta dar clique duplo no arquivo *eclipse.exe* e a ferramenta irá iniciar.

Por fim, precisamos selecionar também os jars do Selenium para termos pleno suporte ao mesmo dentro dos projetos. Optaremos por usar o Java como linguagem server side, em vista de a imensa maioria dos usuários do Selenium usarem essa linguagem no mercado por padrão. Portanto, acesse a página de download do Selenium (seção **Links**) e na seção “*Selenium Client & WebDriver Language Bindings*” clique no link correspondente à linguagem Java (**Figura 1**).

Selenium Client & WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Selenium RC, Selenium Remote WebDriver) or create local Selenium WebDriver script you need to make use of language-specific client drivers. These languages include both 1.x and 2.x style clients.

While language bindings for [other languages exist](#), these are the core ones that are supported by the main project hosted on google code.

Language	Client Version	Release Date	Download	Change log	Javadoc
Java	2.46.0	2015-06-04	Download	Change log	Javadoc
C#	2.46.0	2015-06-04	Download	Change log	API docs
Ruby	2.46.1	2015-06-04	Download	Change log	API docs
Python	2.46.0	2015-06-04	Download	Change log	API docs
Javascript (Node)	2.45.0	2015-02-26	Download	Change log	API docs

Figura 1. Página de download do WebDriver do Selenium

Nome	Data de modificação	Tipo	Tamanho
libs	09/06/2015 16:14	Pasta de arquivos	
CHANGELOG	04/06/2015 12:17	Arquivo	82 KB
LICENSE	04/06/2015 12:17	Arquivo	12 KB
NOTICE	04/06/2015 12:17	Arquivo	1 KB
selenium-java-2.46.0.jar	04/06/2015 12:17	Arquivo JAR	3.816 KB
selenium-java-2.46.0-sources.jar	04/06/2015 12:17	Arquivo JAR	655 KB

Figura 2. Estrutura de pastas e arquivos do Selenium

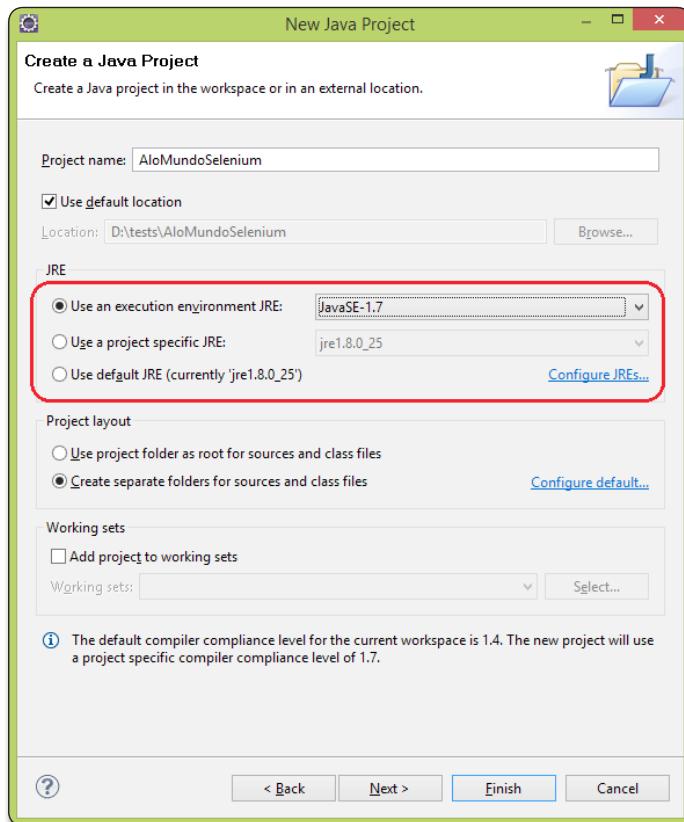


Figura 3. Wizard de criação de novo projeto Selenium no Eclipse

O arquivo baixado estará também no formato zip, logo extraia-o novamente no diretório do projeto. A **Figura 2** mostra a representação dos arquivos internos, dos quais faremos uso essencialmente dos jars.

Execute agora o Eclipse e selecione um workspace para salvar o projeto. Após isso, vamos criar um primeiro exemplo de Alo-Mundo com o browser Firefox. Selecione a opção “File > New > Project > Java Project” e, na janela que aparecer, dê o nome “AloMundoSelenium” ao projeto e certifique-se de selecionar a opção JavaSE-7 no botão “Use na execution environment JRE”, conforme mostra a **Figura 3**.

Clique em “Finish”. Para que as nossas classes reconheçam o Selenium WebDriver, precisamos adicionar os jars ao classpath. Para isso, clique com o botão direito no projeto gerado e selecione “Properties > Java Build Path” e acesse a tab “Libraries”. Clique no

botão “Add External JARS...” e navegue até as pastas onde você descompactou o Selenium. Selecione todos os jars da raiz do diretório e da pasta “libs” e clique em “Abrir” (**Figura 4**).

Para fazer um teste rápido, vá até o diretório src do projeto, clique com o botão direito e selecione a opção “New > Class” e dê um nome de pacote (br.edu.devmedia.selenium) e um nome de classe “AloMundoSelenium” e clique em “Finish”. Na classe que for aberta, adicione o conteúdo da **Listagem 1** à mesma.

Listagem 1. Classe de teste com o browser Firefox.

```
01 package br.edu.devmedia.selenium;
02
03 import java.io.File;
04
05 import org.openqa.selenium.WebDriver;
06 import org.openqa.selenium.chrome.ChromeDriver;
07 import org.openqa.selenium.firefox.FirefoxDriver;
08 import org.openqa.selenium.ie.InternetExplorerDriver;
09
10 public class AloMundoSelenium {
11     public static void main(String[] args) {
12         abrirFirefox();
13     }
14
15     private static void abrirFirefox() {
16         WebDriver driver = new FirefoxDriver();
17         driver.get("http://devmedia.com.br");
18         String i = driver.getCurrentUrl();
19         System.out.println(i);
20         driver.close();
21     }
22
23 }
```

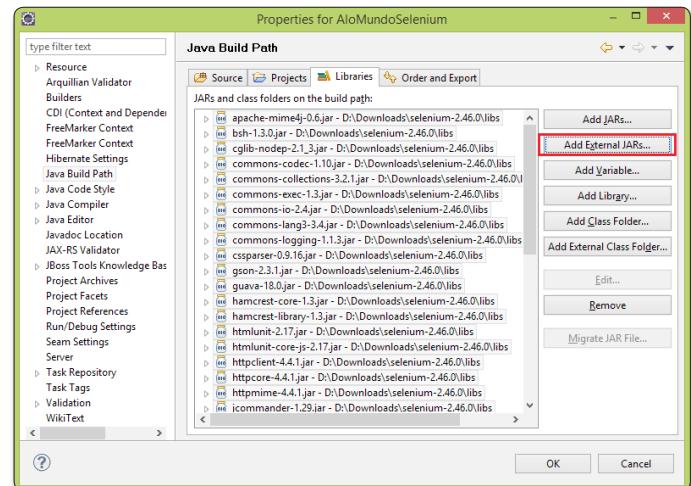


Figura 4. Libs do Selenium adicionadas ao projeto

Testes automatizados com o framework Selenium

O código traz uma representação simples de teste no Selenium que irá somente abrir uma instância do browser Firefox (linha 16), acessar o site da DevMedia (linha 17) e retornar a URL para o qual o mesmo foi redirecionado (linha 18). No final, ele fecha a conexão com o driver (linha 20). Para executar vá até o menu “Run” no topo da página ou clique com o botão direito na classe e selecione “Run As > Java Application”.

O resultado será a abertura de uma instância do browser citado com o respectivo site e a mensagem final impressa no console (**Figura 5**).

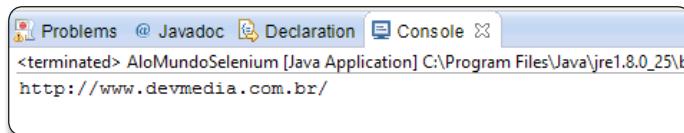


Figura 5. Resultado de execução do teste no Firefox

O Selenium trabalha com o conceito de drivers para o browser a ser trabalhado. Nos projetos reais, geralmente temos código para lidar com ao menos três browsers: Firefox, Chrome e IE, que são os mais usados. Caso você queria adicionar uma nova instância, precisará também efetuar o download do driver dele, uma vez que o Selenium reconhece apenas o Firefox em vista da sua IDE ser desenvolvida nele. Para isso, efetue o download do arquivo chromedriver.exe na sua versão mais recente no endereço disponível na seção **Links** e posicione-o no endereço do workspace do projeto.

O código para acessar agora passa a ser igual ao da **Listagem 2**.

Veja que na linha 6 temos a importação do arquivo do driver diretamente da pasta onde ele foi alocado e, em seguida, setamos a propriedade “webdriver.chrome.driver” nas propriedades do sistema com o valor do caminho do driver, para que o Selenium possa encontrar uma instância correta da classe ChromeDriver. O restante do procedimento é o mesmo. Na linha 25 vemos o equivalente para o browser do Internet Explorer, mas também será necessário baixar o driver do mesmo.

Um outro bom exemplo é quando queremos não só acessar uma página, como também enviar informações para ela, gerar submits e recuperar dados dos responses. Esse tipo de funcionalidade é possível através da classe HTMLUnitDriver, que funciona como os mesmos drivers que vimos para os browsers, porém trabalhando internamente, sem interface gráfica. Essa estratégia é muito usada por desenvolvedores ou testadores que querem criar scripts rapidamente sem ter de ficar acessando o browser o tempo todo.

Vamos criar um exemplo de acesso à página do Google e enviar uma palavra para ser pesquisada, dando submit no formulário e recebendo a resposta. Quando uma pesquisa é feita o título da página do Google muda de acordo com o texto, assim podemos recuperar essa nova informação e checar se o teste funcionou. Veja a **Listagem 3** para isso.

Agora é possível ver o acesso direto ao browser virtual do HTMLUnitDriver que também pode ser retornado para uma instância do WebDriver (linha 13). O método findElement() da linha 17 recupera o conteúdo HTML completo da página e faz

Listagem 2. Código para acessar a instância do Chrome.

```
01 package br.edu.devmedia.selenium;
02
03 import java.io.File;
04
05 import org.openqa.selenium.WebDriver;
06 import org.openqa.selenium.chrome.ChromeDriver;
07 import org.openqa.selenium.firefox.FirefoxDriver;
08 import org.openqa.selenium.ie.InternetExplorerDriver;
09
10 public class AloMundoSelenium {
11     public static void main(String[] args) {
12         abrirChrome();
13     }
14
15     private static void abrirChrome() {
16         File file = new File("D:/Downloads/chromedriver.exe");
17         System.setProperty("webdriver.chrome.driver", file.getAbsolutePath());
18         WebDriver driver = new ChromeDriver();
19         driver.get("http://devmedia.com.br");
20         String i = driver.getCurrentUrl();
21         System.out.println(i);
22         driver.close();
23     }
24
25     private static void abrirIE() {
26         WebDriver driver = new InternetExplorerDriver();
27         driver.get("http://devmedia.com.br");
28         String i = driver.getCurrentUrl();
29         System.out.println(i);
30         driver.close();
31     }
32 }
```

Listagem 3. Código de teste para pesquisa no Google.

```
01 package br.edu.devmedia.selenium;
02
03 import java.util.concurrent.TimeUnit;
04
05 import org.openqa.selenium.By;
06 import org.openqa.selenium.JavascriptExecutor;
07 import org.openqa.selenium.WebDriver;
08 import org.openqa.selenium.WebElement;
09 import org.openqa.selenium.htmlunit.HtmlUnitDriver;
10
11 public class HTMLDriver {
12     public static void main(String[] args) {
13         WebDriver driver = new HtmlUnitDriver();
14
15         driver.get("http://www.google.com");
16
17         WebElement element = driver.findElement(By.name("q"));
18
19         element.sendKeys("Queijo!");
20
21         element.submit();
22
23         System.out.println("O título da página é: " + driver.getTitle());
24
25         driver.quit();
26     }
27 }
```

uma busca pelo critério definido no se parâmetro que, por sua vez, será responsável por buscar um elemento único de nome “q” (no caso, a caixa de pesquisa da página do Google). Os critérios podem ser vários como por nome, id, classe CSS, etc.

O método `sendKeys()` na linha 19 envia o valor exatamente para o campo de input que deve ser preenchido antes do submit (linha 21) ser enviado. No final, extraímos o valor do atributo “title” do mesmo driver que já estará atualizado com a resposta.

Para conferir o resultado, execute a aplicação via comando `Run` e você verá algo parecido com o conteúdo da **Listagem 4**.

Listagem 4. Código de resultado da execução.

```
01 ADVERTÊNCIA: CSS error:'http://www.google.com.br/?gfe_rd=cr&ei=Z1h4VdzSMabX8gew3oHgAg'[1:9364] Error in expression;
  'found after identifier "progid".
02 jun 10, 2015 12:31:42 PM com.gargoylesoftware.htmlunit.DefaultCssError
  Handler error
03 [...]
04 ADVERTÊNCIA: CSS error:'http://www.google.com.br/?gfe_rd=cr&ei=Z1h4VdzSMabX8gew3oHgAg'[1:12791] Error in expression. (Invalid token ";".
  Was expecting one of: <S>, <NUMBER>, "inherit", <IDENT>, <STRING>, "-", <PLUS>, <HASH>, <EMS>, <EXS>, <LENGTH_px>, <LENGTH_cm>, <LENGTH_mm>, <LENGTH_in>, <LENGTH_pt>, <LENGTH_pc>, <ANGLE_deg>, <ANGLE_rad>, <ANGLE_grad>, <TIME_ms>, <TIME_s>, <FREQ_hz>, <FREQ_khz>, <RESOLUTION_dpi>, <RESOLUTION_dpcm>, <PERCENTAGE>, <DIMENSION>, <URI>, <FUNCTION>.)
05 [...]
06 jun 10, 2015 12:31:42 PM com.gargoylesoftware.htmlunit.DefaultCssError
  Handler error
07 ADVERTÊNCIA: CSS error:'http://www.google.com.br/?gfe_rd=cr&ei=Z1h4VdzSMabX8gew3oHgAg'[1:15068] Error in expression;
  'found after identifier "progid".
08 jun 10, 2015 12:31:42 PM com.gargoylesoftware.htmlunit.DefaultCssError
  Handler error
09 [...]
10 O título da página é: Queijo! - Pesquisa Google
```

Veja que no final da listagem temos a exibição do título da página. Nas linhas anteriores temos alguns erros exibidos (entre outros que foram omitidos da listagem) que caracterizam o processo de validação do Selenium, ou seja, sempre que alguma tag não for fechada ou faltar algum atributo obrigatório ou tivermos qualquer erro a nível de HTML, essas linhas serão impressas. Mas não precisa se preocupar com isso, esse é um comportamento padrão do Selenium.

Em algumas situações será preciso também acessar propriedades via JavaScript de dentro das páginas. Por exemplo, suponha que em vez de acessarmos o título da página via método `getTitle()` do driver, nós quiséssemos executar um script JavaScript que fizesse o mesmo procedimento. Para isso, precisaríamos fazer uso da classe `JavaScriptExecutor` da API do Selenium e passar a string do nosso script como parâmetro para o seu método `executeScript()`.

Entretanto, como em todo browser, também precisamos habilitar o seu suporte ao JavaScript. Os browsers comuns já vêm com ele habilitado, mas não é o caso do `HTMLUnitDriver` do Selenium. Vejamos o código da **Listagem 5** para isso. Vamos acessar agora

o site da DevMedia que tem mais recursos visuais para usarmos de exemplo.

Listagem 5. Código para busca de elementos via JavaScript.

```
01 package br.edu.devmedia.selenium;
02
03 import java.util.concurrent.TimeUnit;
04
05 import org.openqa.selenium.By;
06 import org.openqa.selenium.JavascriptExecutor;
07 import org.openqa.selenium.WebDriver;
08 import org.openqa.selenium.WebElement;
09 import org.openqa.selenium.htmlunit.HtmlUnitDriver;
10
11 public class HTMLDriver {
12   public static void main(String[] args) {
13     HtmlUnitDriver driver = new HtmlUnitDriver();
14     driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
15     driver.setJavascriptEnabled(true);
16
17     driver.get("http://www.devmedia.com.br/");
18
19     WebElement element = driver.findElementByCssSelector("dm-busca");
20     System.out.println("Cor de fundo:" + element.getCssValue("background-color"));
21     System.out.println("Padding:" + element.getCssValue("padding"));
22     System.out.println("Classe CSS:" + element.getAttribute("class"));
23
24     JavascriptExecutor javascript = (JavascriptExecutor) driver;
25     String tituloPagina = (String) javascript.executeScript(
26       "return document.title");
27
28     System.out.println("O título da página é :" + tituloPagina);
29   }
30 }
```

Veja que na linha 14 estamos configurando um timeout para as requisições feitas ao `HTMLUnitDriver`. Isso porque os testes podem ser impactados quando um request demorar muito a retornar, logo você tem essa opção em mãos semelhante ao que fazemos com Web Services. O método `implicitlyWait()` define o tempo máximo de processamento, no caso cinco segundos.

Em seguida, na linha 15, usamos o método `setJavascriptEnabled()` para definir que o JavaScript estará habilitado nesse browser. Se estiver usando uma instância do Firefox ou Chrome como fizemos antes, esse código não será necessário.

Nas linhas seguintes vemos algumas propriedades novas, a saber:

- Linha 19: O método `findElementByCssSelector()` recebe o seletor do elemento a ser recuperado na página HTML, no caso a caixa de pesquisas da página inicial; A **Tabela 1** traz uma lista completa dos métodos possíveis.
- Linhas 20 a 22: Imprimimos o valor das propriedades CSS de cor de fundo, padding e classe do mesmo elemento, respectivamente;
- Linha 24: Instanciamos um novo objeto do tipo `JavascriptExecutor` a partir do próprio objeto `driver` (que o herda automaticamente pela herança de classes);
- Linha 25: Chamamos o método `executeScript()` passando o JavaScript que queremos que seja executado no browser.

Testes automatizados com o framework Selenium

Essa cláusula sempre deve vir precedida da palavra-chave “return”, já que precisamos que algo seja retornado;

- Linha 28: Encerramos o driver.

Para verificar o resultado, reexecute a aplicação e você deverá ver no console a mensagem impressa pela **Listagem 6**.

Listagem 6. Mensagem final de execução do teste.

```
Cor de fundo: #f0f0f0  
Padding: 95px 0 45px 0  
Classe CSS: container-fluid dm-busca  
O título da página é: DevMedia - Tutoriais, Videos e Cursos de Programação
```

Agora suponha que precisamos migrar toda essa lógica de testes para o Firefox. Mais que isso, precisamos enviar apenas um texto de sugestão para a caixa de pesquisa do Google, mas sem clicar no submit. E após isso, esperar até que a caixa de sugestões apareça para que recuperemos todas as sugestões que o Google deu para a palavra informada. Vejamos como ficaria o nosso método main (**Listagem 7**).

Até a linha 18 fazemos a mesma configuração de recuperação do campo de pesquisa e informação da string a ser pesquisada. Na linha 20 criamos uma variável inteira com um valor cinco segundos à frente do tempo atual através do método currentTimeMillis(), assim teremos como pausar a execução do Selenium até que a página de sugestões seja gerada. O teste se baseia nos cinco segundos ou até que o objeto driver encontre o elemento de classe CSS “sbdd_b” que corresponde à respectiva caixa.

Uma vez encontrada (linha 24), recuperamos todos os elementos que casem com o xpath (expressão de caminho) da linha 29. Essa expressão diz que o driver tem de buscar por todos os elementos do tipo div que tenham classe CSS com valor “sbqs_c” que correspondem aos itens de sugestão.

No final, iteramos sobre essa lista (linha 31) e exibimos todos os seus valores (geralmente quatro). O interessante a se notar é que dessa vez acompanharemos a execução já que estamos executando via driver do Firefox.

Para ver o resultado, reexecute a aplicação e aguarde. Você deverá ver uma tela semelhante à da **Figura 6** aparecer e fechar sozinha quando finalizar o teste, bem como o texto da **Listagem 8** aparecer no console da IDE.

Listagem 7. Teste caixa de sugestões do Google.

```
01 package br.edu.devmedia.selenium;  
02  
03 import java.util.concurrent.TimeUnit;  
04  
05 import org.openqa.selenium.By;  
06 import org.openqa.selenium.JavascriptExecutor;  
07 import org.openqa.selenium.WebDriver;  
08 import org.openqa.selenium.WebElement;  
09 import org.openqa.selenium.htmlunit.HtmlUnitDriver;  
10  
11 public class HTMLDriver {  
12     public static void main(String[] args) throws Exception {  
13         WebDriver driver = new FirefoxDriver();  
14  
15         driver.get("http://www.google.com/webhp?complete=1&hl=en");  
16  
17         WebElement query = driver.findElement(By.name("q"));  
18         query.sendKeys("devmedia");  
19  
20         long end = System.currentTimeMillis() + 5000;  
21         while (System.currentTimeMillis() < end) {  
22             WebElement resultsDiv = driver.findElement(By.className("sbdd_b"));  
23  
24             if (resultsDiv.isDisplayed()) {  
25                 break;  
26             }  
27         }  
28  
29         List<WebElement> allSuggestions = driver.findElements(  
30             By.xpath("//div[@class='sbqs_c']"));  
31  
32         for (WebElement suggestion : allSuggestions) {  
33             System.out.println(suggestion.getText());  
34         }  
35         driver.quit();  
36     }  
37 }
```

Listagem 8. Mensagem final de execução do teste com sugestões impressas.

```
devmedia  
devmedia cursos  
devmedia player  
devmedia java magazine  
jun 10, 2015 1:04:47 PM org.openqa.selenium.os.UnixProcess$SeleniumWatchDog  
destroyHarder  
INFORMAÇÕES: Command failed to close cleanly. Destroying forcefully (v2). org.  
openqa.selenium.os.UnixProcess$SeleniumWatchDog@53aad5d
```

Estratégia	Sintaxe	Descrição
Por ID	driver.findElement(By.id(<ID do elemento>))	Recupera um elemento pelo atributo “id”
Por nome	driver.findElement(By.name(<Nome do elemento>))	Recupera um elemento pelo atributo “name”
Por nome da classe	driver.findElement(By.className(<Classe CSS do elemento>))	Recupera um elemento pelo atributo “class”
Pelo nome da tag	driver.findElement(By.tagName(<Nome da tag HTML>))	Recupera um elemento pelo nome da tag HTML
Pelo texto do link	driver.findElement(By.linkText(<Texto do Link>))	Recupera um link usando seu texto.
Pelo texto parcial do link	driver.findElement(By.partialLinkText(<Texto parcial do Link>))	Recupera um link pelo seu texto parcial
Por CSS	driver.findElement(By.cssSelector(<Seletor do elemento>))	Recupera um elemento pelo seu seletor CSS
Por XPath	driver.findElement(By.xpath(<expressão query xpath>))	Recupera um elemento usando um query XPath

Tabela 1. Lista de métodos para recuperar elementos no Selenium



Figura 6. Tela de teste no Firefox com sugestões

A mensagem final quer dizer que, uma vez que chamamos o método driver.quit() a instância do Firefox será finalizada.

Seletores jQuery

A biblioteca jQuery é extremamente mais simples e produtiva de usar em relação ao JavaScript puro. Você pode optar por usá-la em conjunto com seus seletores para recuperar elementos, definir eventos e funções de callback, bem como enviar submits em formulários. Para criar um cenário mais próximo da realidade dos projetos, vamos criar uma página de cadastro de usuário simples com suporte a jQuery para os scripts e Bootstrap para o estilo e estrutura da mesma. Vejamos na **Listagem 9** o código para isso.

A listagem faz a importação no início dos arquivos de script necessários para carregar o jQuery e o estilo do Bootstrap (linhas 7 a 9). Os imports foram feitos via CDN (*Content Delivery Network*) portanto,

não precisamos baixá-los estaticamente. A página traz apenas dois campos de input (email e senha, linhas 22 e 32), três checkboxes, dois botões de rádio e o botão de submit no fim da mesma. Você pode visualizar a página executando o arquivo diretamente no browser, tal como mostrado na **Figura 7**.

Figura 7. Tela de cadastro de usuários para teste

Listagem 9. Código da página HTML de cadastro de usuários.

```

01 <!DOCTYPE html>
02 <html lang="en">
03 <head>
04   <title>Selenium Teste</title>
05   <meta charset="utf-8">
06   <meta name="viewport" content="width=device-width, initial-scale=1">
07   <link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/
      css/bootstrap.min.css">
08   <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">
      </script>
09   <script src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/js/
      bootstrap.min.js"></script>
10 </head>
11 <body>
12
13 <div class="container">
14   <div class="col-md-12">
15     <h2>Cadastro de Pessoa</h2>
16     <div class="panel panel-info">
17       <div class="panel-heading">Formulário Básico</div>
18       <div class="panel-body">
19         <form role="form">
20           <div class="form-group">
21             <label class="control-label" for="exampleInputEmail1">Email</label>
22             <input type="email" class="form-control" id="exampleInputEmail1"
                  placeholder="Digite seu email">
23           </div>
24
25           <div class="form-group">
26             <label class="control-label" for="exampleInputPassword1">Password</label>
27             <input type="password" class="form-control" id="exampleInputPassword1"
                  placeholder="Digite sua senha">
28           </div>
29
30           <div class="form-group">
31             <div class="checkbox">
32               <label> <input type="checkbox" checked=""> Ativo </label>
33             </div>
34             <p class="help-block">Lorem ipsum dolor sit amet</p>
35           </div>
36
37           <div class="form-group">
38             <label class="control-label">Opções</label>
39             <div class="checkbox">
40               <label> <input type="checkbox" name="optionsRadios"
                     id="optionsCheckbox1" value="option1" checked=""> Opção 1 </label>
41             </div>
42             <div class="checkbox">
43               <label> <input type="checkbox" name="optionsRadios"
                     id="optionsCheckbox2" value="option2"> Opção 2 </label> </div>
44           </div>
45
46           <div class="form-group">
47             <label class="control-label">Sexo</label>
48             <div class="radio">
49               <label> <input type="radio" name="optionsRadios" id="optionsRadios1"
                     value="option1" checked=""> Masculino </label>
50             </div>
51             <div class="radio">
52               <label> <input type="radio" name="optionsRadios" id="optionsRadios2"
                     value="option2"> Feminino </label>
53             </div>
54           </div>
55
56           <button type="submit" class="btn btn-default">Criar Usuário</button>
57         </form>
58       </div>
59     </div></div></div></body></html>
```

Testes automatizados com o framework Selenium

Listagem 10. Código para acessar propriedades da página de cadastro.

```
01 package br.edu.devmedia.selenium;
02
03 import java.util.concurrent.TimeUnit;
04
05 import org.openqa.selenium.By;
06 import org.openqa.selenium.JavascriptExecutor;
07 import org.openqa.selenium.WebDriver;
08 import org.openqa.selenium.WebElement;
09 import org.openqa.selenium.htmlunit.HtmlUnitDriver;
10
11 public class HTMLDriver {
12     public static void main(String[] args) {
13         WebDriver driver = new FirefoxDriver();
14         driver.get("file:///D:/Downloads/login.html");
15
16         List<String> checked = Arrays.asList(new String[]{"optionsCheckbox1",
17             "optionsRadios1"});
18         JavascriptExecutor js = (JavascriptExecutor) driver;
19
20         @SuppressWarnings("unchecked")
21         List<WebElement> checkedElements = (List<WebElement>)
22             js.executeScript("return jQuery.find(':checked')");
23
24         @SuppressWarnings("unchecked")
25         List<WebElement> focusElements = (List<WebElement>)
26             js.executeScript("return jQuery.find(':focus')");
27
28         if (checkedElements.size() == 2)
29             System.out.println("Duas checkboxes selecionadas");
30
31         if (focusElements.size() == 1)
32             System.out.println("Campo com foco: " + focusElements.get(0).getAttribute("id"));
33
34         for (WebElement element : checkedElements) {
35             if (checked.contains(element.getAttribute("id")))
36                 System.out.println("Elemento contém id");
37
38         js.executeScript("jQuery('#optionsRadios2').attr('checked','checked')");
39         js.executeScript("jQuery('button[type='submit']").text('Texto Mudado')");
40
41     }
42 }
```

Nossos objetivos com esta página serão (**Listagem 10**):

- Buscar todos os elementos checked e exibir a quantidade;
- Buscar o id do elemento que recebeu o foco (email);
- Mudar a seleção do campo sexo para feminino;
- Mudar o texto do botão de submit para “Texto Mudado”.

A primeira mudança significativa está na linha 14, onde estamos acessando o arquivo HTML localmente, diretamente da pasta onde você salvou a página. Dessa forma não precisamos hospedar a aplicação para fazer os testes e o Selenium provê esse recurso de maneira off-line. Na linha 16 criamos uma lista de WebElement's para recuperar todos os itens checados (rádio e checkbox) da página. Veja que é nesse trecho onde executamos o primeiro código jQuery que precisa ter o seletor informado dentro método executeScript().

Na linha 23 fazemos a mesma seleção, só que agora para o campo que tem o foco (:focus). Na linha 25 testamos o tamanho da lista de elementos checados e na linha 28 para a lista de elementos com foco. Da linha 31 a 35 verificamos se algum dos elementos recuperados contém o atributo “id”, apenas para imprimir a mensagem de sucesso.

No final, na linha 37 executamos o código jQuery para selecionar o rádio de Feminino, e na linha 38 mudamos o texto do botão de submit.

Para verificar o resultado, reexecute a aplicação e você verá algo semelhante ao que temos na **Figuras 8** (no browser Firefox) e **9** (no console da IDE).

O leitor ainda pode adicionar mais arquivos de scripts ou CSS dinamicamente a partir do código Selenium via variável *var headID* que está disponível para representar o cabeçalho da página. Isso vale, inclusive, para quaisquer frameworks jQuery que queira acrescentar na implementação.

Cadastro de Pessoa

The screenshot shows a user registration form titled 'Cadastro de Pessoa'. It has a 'Formulário Básico' section with fields for 'Email' (placeholder 'Digite seu email') and 'Password' (placeholder 'Digite sua senha'). There is a checkbox labeled 'Ativo' which is unchecked. Below the form, there is a text area containing placeholder text 'Lorem ipsum dolor sit amet'. Under the text area, there is a section titled 'Opções' with two checkboxes: 'Opção 1' (checked) and 'Opção 2'. Below that is a 'Sexo' section with two radio buttons: 'Masculino' (unchecked) and 'Feminino' (checked). At the bottom right of the form is a button labeled 'Texto Mudado'.

Figura 8. Tela de cadastro de usuários após o teste

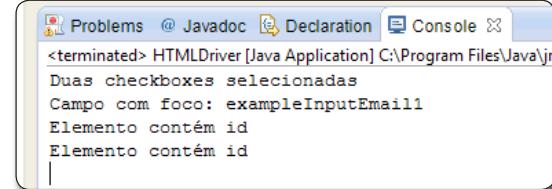


Figura 9. Tela de console com resultado da execução

Selenium IDE na prática

A Selenium IDE é uma ferramenta de testes para aplicações web totalmente integrada com os conceitos do framework no que se refere à gravação e playback dos testes de regressão. Através dela, o desenvolvedor não é mais obrigado a usar qualquer linguagem de programação, apesar de poder fazer isso. Também é possível exportar os scripts para quaisquer linguagens suportadas.

Os scripts criados para a aplicação de teste são chamados de casos de teste e um conjunto de casos de teste é chamado de suíte. Para trabalhar com ela, é preciso primeiro fazer o

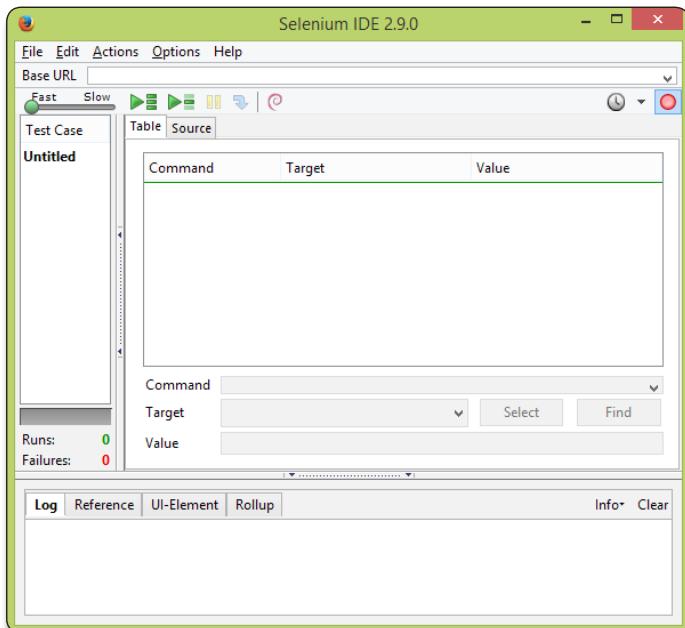


Figura 10. Janela da Selenium IDE no Firefox

download (seção **Links**). Efetue o download da versão mais recente na subseção “Selenium IDE” e dê um click duplo para iniciar a instalação.

É importante que o leitor considere que essa IDE só é suportada pelo Firefox, logo não é possível configurá-la em outros browsers (caso prefira o Chrome, use a opção com Java mostrada na seção anterior). Reinicie o Firefox e pronto, sua IDE está instalada. Para verificar, vá até a opção “Tools > Selenium IDE” e uma janela igual à **Figura 10** irá aparecer.

Na **Tabela 2** você pode encontrar um detalhamento dos principais botões e funcionalidades fornecidos pela IDE.

Primeiros testes na IDE

Agora que temos a IDE instalada, podemos gravar um teste automatizado apenas usando a aplicação desejada normalmente. Façamos um teste com os seguintes passos:

1. Entrar no site da DevMedia.
2. Clicar no botão “Login”.
3. Digitar um login e senha qualquer.
4. Clicar em “OK”.

O teste é demasiadamente simples, mas é o suficiente para que você entenda como o Selenium salva seus scripts. Agora, abra uma nova aba do browser (em branco) e na IDE clique no botão

Botão	Descrição
Controlador de Velocidade	
Botão “Run All”	
Botões “Pause/Resume”	
Botão “Step”	
Botões “Record/Stop Recording”	
Botão “Toggle Schedule”	
Janela “Table”	<p></p> <p>Exibe os comandos gravados. Você pode modificar e incluir novos comandos, que são divididos em três partes: Command: descreve o nome do comando; Target: descreve o elemento alvo (id ou name) do Xpath; Value: descreve o valor do elemento.</p>

Tabela 2. Lista de botões e comandos padrão da Selenium IDE

Testes automatizados com o framework Selenium

“Record”. Em seguida, efetue a lista de passos que mostramos e ao finalizar, clique no botão “Stop Recording”.

Ao final, sua IDE estará semelhante à que temos na **Figura 11**. Veja que temos cinco comandos com tipos e targets diferentes. Por exemplo: o tipo “open” define que entramos numa URL (exibida no topo da janela, no campo “Base URL”); o tipo “click” define que clicamos num elemento de nome “usuario” e o tipo “clickAndWait” define que foi enviado um submit e que o Selenium deve aguardar até que a resposta retorne. Em relação aos campos que precisam de um valor a ser informado (input text), a terceira coluna é preenchida com os mesmos.

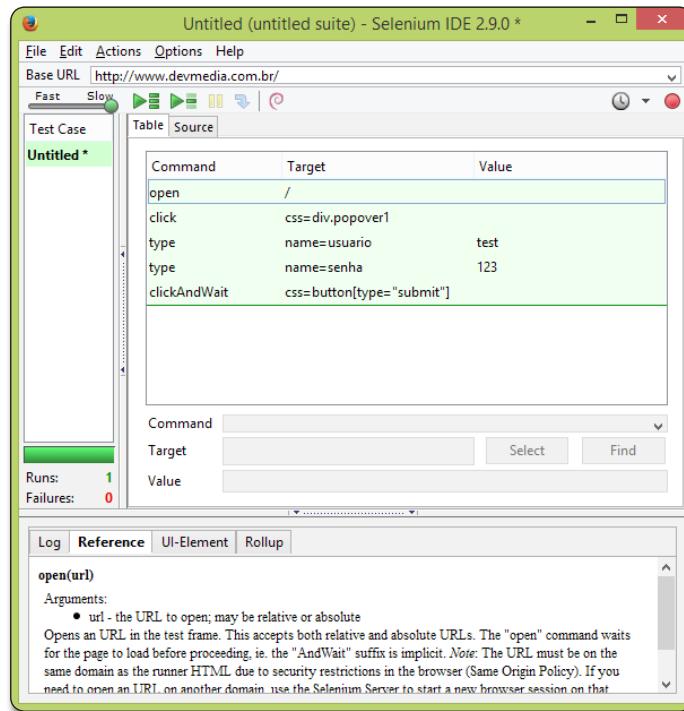


Figura 11. Selenium IDE com testes finalizados e gravados

Você também pode modificar os seus testes via script. Clique na aba “Source” ao lado de “Table” e você verá um conteúdo semelhante ao da **Listagem 11**.

O conteúdo do script é sempre gerado em HTML, logo temos as importações das tags meta e link na tag head, bem como a tabela com os comandos logo abaixo. É aconselhável que o leitor sempre utilize a opção gráfica para evitar qualquer erro de sintaxe na ferramenta.

Agora, sempre que quiser reexecutar o mesmo teste, basta selecionar o nome do *test case* na barra lateral esquerda e clicar no botão “Play test suite”.

Sobre os Casos de Teste

A primeira vez que abrimos a IDE do Selenium o botão de gravação já vem ativado, e isso permite uma interação via uso do site que terá seus scripts salvos. Se você não deseja que a IDE faça isso, vá até o menu “Options > Options...” e desmarque a opção “Start recording immediately on open”.

Listagem 11. Código script gerado para o teste na Selenium IDE.

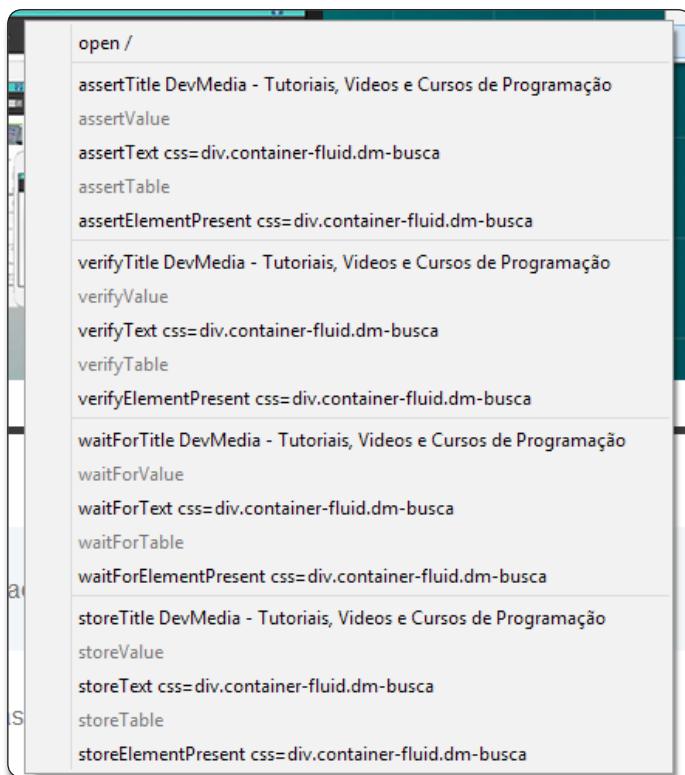
```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
04 <head profile="http://selenium-ide.openqa.org/profiles/test-case">
05 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
06 <link rel="selenium.base" href="http://www.devmedia.com.br/" />
07 <title>New Test</title>
08 </head>
09 <body>
10 <table cellpadding="1" cellspacing="1" border="1">
11 <thead>
12 <tr><td rowspan="1" colspan="3" style="text-align: center;">New Test
```

Quando se está construindo um caso de testes, algumas considerações devem ser levadas em conta:

- Para gravar ações do tipo submit, é necessário que o usuário clique no botão em vez de dar um enter em algum dos campos do formulário. Isso porque o Selenium não consegue reconhecer esse evento e associar a um envio de request, já que o usuário pode dar vários enters, ou ter um controle de navegação de campo para campo via tecla Enter, em vez de Tab.
- Quando clicamos em um link durante uma gravação, o Selenium grava o comando no tipo “click”. Porém, você deve mudar esse tipo para “clickAndWait” para garantir que o seu caso de teste fará uma pausa até que a página carregue por completo e o fluxo possa prosseguir sem erros. Caso contrário, nosso caso de teste continuará executando os comandos antes que a página tenha todos os elementos UI carregados na HTML.

Os seus casos de teste também terão que verificar as propriedades de uma página da web. Isso requer o uso dos comandos *assert*

(comparar dois valores) e *verify* (fazer um teste qualquer). Quando o Selenium IDE estiver gravando, vá até o navegador exibindo seu aplicativo de teste e clique com o botão direito em qualquer lugar da página. Será exibido um menu de contexto mostrando os referidos comandos bem como outros (**Figura 12**) que podem ser usados via Selenium.



```
open /  
assertTitle DevMedia - Tutoriais, Vídeos e Cursos de Programação  
assertValue  
assertText css=div.container-fluid.dm-busca  
assertTable  
assertElementPresent css=div.container-fluid.dm-busca  
verifyTitle DevMedia - Tutoriais, Vídeos e Cursos de Programação  
verifyValue  
verifyText css=div.container-fluid.dm-busca  
verifyTable  
verifyElementPresent css=div.container-fluid.dm-busca  
waitForTitle DevMedia - Tutoriais, Vídeos e Cursos de Programação  
waitForValue  
waitForText css=div.container-fluid.dm-busca  
waitForTable  
waitForElementPresent css=div.container-fluid.dm-busca  
storeTitle DevMedia - Tutoriais, Vídeos e Cursos de Programação  
storeValue  
storeText css=div.container-fluid.dm-busca  
storeTable  
storeElementPresent css=div.container-fluid.dm-busca
```

Figura 12. Menu de contexto com comandos Selenium

A primeira vez que você acessar a IDE, terá apenas um ou dois comandos disponíveis. Mas à medida que for criando seus casos de teste e o browser for efetuando operações mais complexas, como submits, scrolling, dentre outras, o Selenium irá preenchendo essa lista automaticamente com as opções que ela julgar que você possa precisar. Dentre a lista de comandos mostrada na figura, os principais são:

- **open:** abre uma página usando a URL passada;
- **click/clickAndWait:** executa uma operação de clique e, opcionalmente, aguarda uma nova página carregar;
- **verifyTitle/assertTitle:** verifica se um título esperado para a página está presente;
- **verifyTextPresent:** verifica se um texto esperado está em algum lugar na página;
- **verifyElementPresent:** verifica se um elemento UI esperado, tal como definido pela sua tag HTML, está presente na página;

- **verifyText:** verifica se o texto esperado e sua tag HTML correspondente estão presentes na página;
- **verifyTable:** verifica se conteúdo esperado de uma tabela está presente;
- **waitForPageToLoad:** finaliza a execução até que uma nova página seja carregada. Chamado automaticamente quando o clickAndWait é usado;
- **waitForElementPresent:** finaliza a execução até que um elemento da interface do usuário esperado, conforme definido pela sua tag HTML, está presente na página.

Existem inúmeras outras abordagens nas quais o leitor poderá usar o Selenium. Como vimos, a automação precisa estar obrigatoriamente vinculada a telas, estejam elas executando num PC, smartphone, tablet, etc. O Selenium conta ainda com dois drivers específicos para ambientes móveis: AndroidDriver, classe da API do Selenium exclusiva para dispositivos Android; e iWebDriver, para simular testes em ambiente iOS.

Também é possível integrar o Selenium com o JUnit, QUnit, JTestDriver e outros inúmeros frameworks de teste unitário disponíveis no mercado. Os arquivos que eles geram em conjunto são leves e podem facilmente se adaptar a um controle de integração com o Jenkins, por exemplo, bem como ser manuseados por ferramentas de versionamento (Git, Mercurial, etc.).

Para os desenvolvedores PHP que querem fazer uso do Selenium, o mesmo não provê suporte nativo à linguagem, mas é possível usar o PHPUnit: uma instância do xUnit para testes unitários que traz consigo módulos do Selenium que podem ser instalados via linha de comando.

Autor



Sueila Sousa

É tester e entusiasta de tecnologias front-end. Atualmente trabalha como analista de testes na empresa Indra, com foco em projetos de desenvolvimento de sistemas web, totalmente baseados em JavaScript e afins. Possui conhecimentos e experiências em áreas como Gerenciamento de processos, banco de dados, além do interesse por tecnologias relacionadas ao desenvolvimento e teste client side.



Links:

Página oficial de download do JDK.

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Página oficial de download do Eclipse.

<https://eclipse.org/downloads/>

Página de download do driver do Chrome para o Selenium.

<http://chromedriver.storage.googleapis.com/index.html>

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