

Vue.js

na prática

Bônus:

-
- Vuex
- Vue Router
- Firebase

Vue.js na prática (PT-BR)

Daniel Schmitz e Daniel Pedrinha Georgii

Esse livro está à venda em <http://leanpub.com/livro-vue>

Essa versão foi publicada em 2017-12-29



Esse é um livro [Leanpub](#). A Leanpub dá poderes aos autores e editores a partir do processo de Publicação Lean. [Publicação Lean](#) é a ação de publicar um ebook em desenvolvimento com ferramentas leves e muitas iterações para conseguir feedbacks dos leitores, pivotar até que você tenha o livro ideal e então conseguir tração.

© 2016 - 2017 Daniel Schmitz e Daniel Pedrinha Georgii

Gostaria de agradecer as fantásticas comunidades brasileiras:

vuejs-brasil.slack.com

laravel-br.slack.com

telegram.me/vuejsbrasil

Aos autores do blog www.vuejs-brasil.com.br por divulgarem o vue em português. Vocês são incríveis!

Conteúdo

Parte 1 - Conhecendo o Vue	1
1. Introdução	2
1.1 Tecnologias empregadas	2
1.2 Instalação do node	4
1.3 Uso do npm	5
1.4 Conhecendo um pouco o RESTfull	6
2. Conhecendo Vue.js	8
2.1 Uso do jsFiddle	8
2.2 Configurando o jsFiddle para o Vue	9
2.3 Hello World, vue	11
2.4 Two way databind	13
2.5 Criando uma lista	14
2.6 Detectando alterações no Array	16
Utilizando v-bind:key	17
Uso do set	18
Como remover um item	18
Loops em objetos	18
2.7 Eventos e métodos	19
Modificando a propagação do evento	20
Modificadores de teclas	21
2.8 Design reativo	22
2.9 Criando uma lista de tarefas	22
2.10 Eventos do ciclo de vida do Vue	27
2.11 Compreendendo melhor o Data Bind	29
Databind único	29

CONTEÚDO

	Databind com html	30
	Databind em Atributos	30
	Expressões	30
2.12	Filtros	31
2.13	Diretivas	31
	Argumentos	32
	Modificadores	32
2.14	Atalhos de diretiva (Shorthands)	32
2.15	Alternando estilos	33
2.16	Uso da condicional v-if	35
2.17	Exibindo ou ocultando um bloco de código	36
2.18	v-if vs v-show	36
2.19	Formulários	36
	Checkbox	37
	Radio	38
	Select	38
	Atributos para input	38
2.20	Conclusão	39

Parte 1 - Conhecendo o Vue

1. Introdução

Seja bem vindo ao mundo Vue (pronuncia-se “view”), um framework baseado em componentes reativos, usado especialmente para criar interfaces web, na maioria das vezes chamadas de SPA - Single Page Application ou aplicações de página única, com somente um arquivo html. Vue.js foi concebido para ser simples, reativo, baseado em componentes, compacto e expansível.

Nesta obra nós estaremos focados na aprendizagem baseada em exemplos práticos, no qual você terá a chance de aprender os conceitos iniciais do framework, e partir para o desenvolvimento de uma aplicação um pouco mais complexa.

1.1 Tecnologias empregadas

Nesta obra usaremos as seguintes tecnologias:

Node

Se você é desenvolvedor Javascript com certeza já conhece o node. Para quem está conhecendo agora, o node pode ser caracterizado como uma forma de executar o Javascript no lado do servidor. Com esta possibilidade, milhares de desenvolvedores criam e publicam aplicações que podem ser usadas pelas comunidade. Graças ao node, o Javascript tornou-se uma linguagem amplamente empregada, ou seria mérito do Javascript possibilitar uma tecnologia como o node? Deixamos a resposta para o leitor.

npm

O node package manager é o gerenciador de pacotes do Node. Com ele pode-se instalar as bibliotecas javascript/css existentes, sem a necessidade de realizar o download do arquivo zip, descompactar e mover para o seu projeto. Com npm também podemos, em questão de segundos, ter uma aplicação base pronta para uso. Usaremos muito o **npm** nesta obra. Se você ainda não a usa, com os exemplos mostrados ao longo do livro você terá uma boa base nessa tecnologia.

Editor de textos

Você pode usar qualquer editor de textos para escrever o seu código Vue. Recomenda-se utilizar um editor leve e que possua suporte ao vue, dentre estes temos:

- Sublime Text 3
- Visual Studio Code
- Atom

Todos os editores tem o plugin para dar suporte ao Vue. Nesta obra usaremos extensivamente o Visual Studio Code.

Servidor Web

Em nossos exemplos mais complexos, precisamos comunicar com o servidor para realizar algumas operações com o banco de dados. Esta operação não pode ser realizada diretamente pelo Javascript no cliente. Temos que usar alguma linguagem no servidor. Nesta obra estaremos utilizando o próprio Node, juntamente com o servidor Express para que possamos criar um simples blog devidamente estruturado.

Browserify

Este pequeno utilitário é um “module bundler” capaz de agrupar vários arquivos javascript em um, possibilitando que possamos dividir a aplicação em vários pacotes separados, sendo agrupados somente quando for necessário. Existe outro *modules bundler* chamado webpack, que é mais complexo consequentemente mais poderoso - que deixaremos de lado nessa obra, não por ser ruim, mas por que o browserify atende em tudo que precisamos.

Material Design e materialize-css

Material Design é um conceito de layout criado pelo Google, usado em suas aplicações, como o Gmail, Imbox, Plus etc. O conceito engloba um padrão de design que pode ser usado para criar aplicações. Como os sistemas web usam folha de estilos (CSS), usamos a biblioteca materialize-css, que usa o conceito do material design.

Postman

Para que possamos testar as requisições REST, iremos fazer uso constante do Postman, um plugin para o Google Chrome que faz requisições ao servidor. O Postman irá simular a requisição como qualquer cliente faria, de forma que o programador que trabalha no lado do servidor não precisa necessariamente programar no lado do cliente.

1.2 Instalação do node

Node e npm são tecnologias que você precisa conhecer. Se ainda não teve a oportunidade de usá-las no desenvolvimento web, esse é o momento. Nesta obra, não iremos abordar a instalação de frameworks javascript sem utilizar o npm.

Para instalar o node/npm no Linux, digite na linha de comando:

```
1 sudo apt-get install git node npm
2 sudo ln -s /usr/bin/nodejs /usr/bin/node
```

Para instalar o Node no Windows, acesse o [site oficial](#)¹ e faça o download da versão estável. Certifique-se de selecionar o item “npm package manager” para instalar o npm também.

Verifique a versão do node no seu ambiente Linux através do comando `node -v`, e caso não seja 5 ou superior, proceda com esta instalação:

```
1 sudo curl -sL https://deb.nodesource.com/setup_6.x | sudo -\ 
2 E bash - 
3 sudo apt-get install -y nodejs
4 sudo ln -s /usr/bin/nodejs /usr/bin/node
```

¹<https://nodejs.org/en/>

1.3 Uso do npm

Será através do *npm* que instalaremos quase todas as ferramentas necessárias para o desenvolvimento. Para compreender melhor como o **npm** funciona, vamos exibir alguns comandos básicos que você irá usar na linha de comando (tanto do Linux, quanto do Windows):

npm init

Este comando inicializa o npm no diretório corrente. Inicializar significa que o arquivo `package.json` será criado, com várias informações sobre o projeto em questão, como o seu nome, a versão do projeto, o proprietário entre outras. Além de propriedades do projeto, também são armazenadas os frameworks e bibliotecas adicionados ao projeto.

npm install ou npm i

Instala uma biblioteca que esteja cadastrada na base do npm, que pode ser acessada [neste endereço](#)². O comando `npm i` produz o mesmo efeito. Quando uma biblioteca é adicionada, o diretório `node_modules` é criado e geralmente a biblioteca é adicionada em `node_modules/nomedabiblioteca`. Por exemplo, para instalar o `vue`, execute o comando `npm i vue` e perceba que o diretório `node_modules/vue` foi adicionado.

npm i -save ou npm i -S

Já sabemos que `npm i` irá instalar uma biblioteca ou framework. Quando usamos `--save` ou `--S` dizemos ao npm para que este framework seja adicionado também ao arquivo de configuração `package.json`. O framework será referenciado no item `dependencies`.

npm i -saveDev ou npm i -D

Possui um comportamento semelhante ao item acima, só que a configuração do pacote instalado é referenciado no item `devDependencies`. Use a opção `-D` para instalar pacotes que geralmente não fazem parte do projeto principal, mas que são necessários para o desenvolvimento tais como testes unitários, automatização de tarefas, um servidor virtual de teste etc.

²<https://www.npmjs.com/>

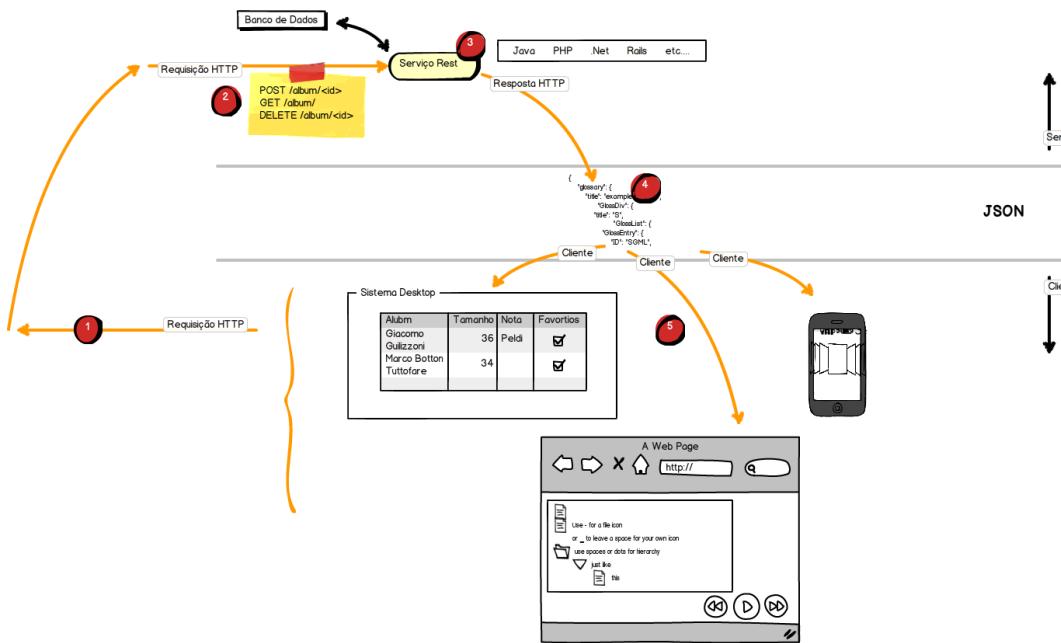
npm i -g

Instala a biblioteca/framework de forma global ao sistema, podendo assim ser utilizado em qualquer projeto. Por exemplo, o `live-server` é um pequeno servidor web que “emula” o diretório atual como um diretório web e cria um endereço para acesso, como `http://localhost:8080`, abrindo o navegador no diretório em questão. Como se usa o `live-server` em quase todos os projetos javascript criados, é comum usar o comando `npm i -g live-server` para que se possa usá-lo em qualquer projeto.

1.4 Conhecendo um pouco o RESTfull

Na evolução do desenvolvimento de sistemas web, os serviços chamados *webservices* estão sendo gradativamente substituídos por outro chamado *RESTful*, que é ‘quase’ a mesma coisa, só que possui um conceito mais simples. Não vamos nos prender em conceitos, mas sim no que importa agora. O que devemos saber é que o Slim Framework vai nos ajudar a criar uma API REST na qual poderemos fazer chamadas através de uma requisição HTTP e obter o resultado em um formato muito mais simples que o XML, que é o JSON.

A figura a seguir ilustra exatamente o porquê do *RESTful* existir. Com ela (e com o slim), provemos um serviço de dados para qualquer tipo de aplicação, seja ela web, desktop ou mobile.



Nesta imagem, temos o ciclo completo de uma aplicação RESTful. Em '1', temos o cliente realizando uma requisição HTTP ao servidor. Todo ciclo começa desta forma, com o cliente requisitando algo. Isso é realizado através de uma requisição http 'normal', da mesma forma que um site requisita informações a um host.

Quando o servidor recebe essa requisição, ele a processa e identifica qual api deve chamar e executar. Nesse ponto, o cliente não mais sabe o que está sendo processado, ele apenas está aguardando a resposta do servidor. Após o processamento, o servidor responde ao cliente em um formato conhecido, como o json. Então, o que temos aqui é o cliente realizando uma consulta ao servidor em um formato conhecido (http) e o servidor respondendo em json.

Desta forma, conseguimos garantir uma importância muito significativa entre servidor e cliente. Ambos não se conhecem, mas sabem se comunicar entre si. Assim, tanto faz o cliente ser um navegador web ou um dispositivo mobile. Ou tanto faz o servidor ser PHP ou Java, pois a forma de conversa entre elas é a mesma.

2. Conhecendo Vue.js

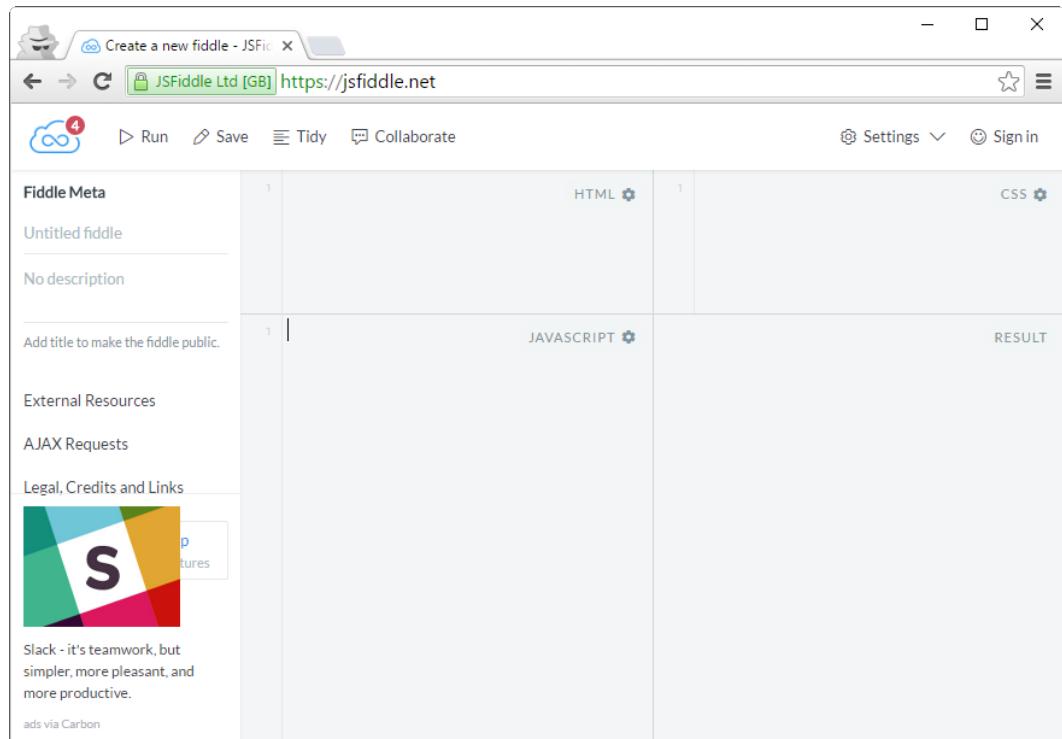
Neste capítulo iremos aprender alguns conceitos básicos sobre o Vue. Não veremos (ainda) a instalação do mesmo, porque como estamos apresentando cada conceito em separado, é melhor usarmos um editor online, neste caso o jsFiddle.

2.1 Uso do jsFiddle

jsFiddle é um editor html/javascript/css online, sendo muito usado para aprendizagem, resolução de problemas rápidos e pequenos testes. É melhor utilizar o jsFiddle para aprendermos alguns conceitos do Vue do que criar um projeto e adicionar vários arquivos.

Acesse no seu navegador o endereço jsfiddle.net¹. Você verá uma tela semelhante a figura a seguir:

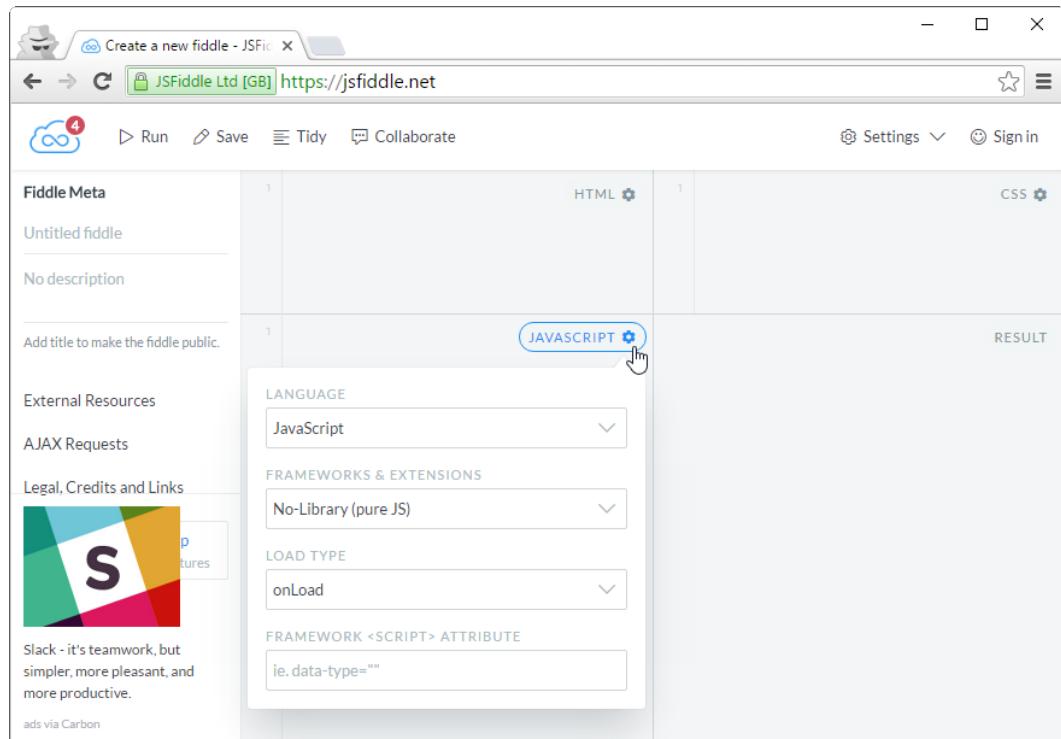
¹jsfiddle.net



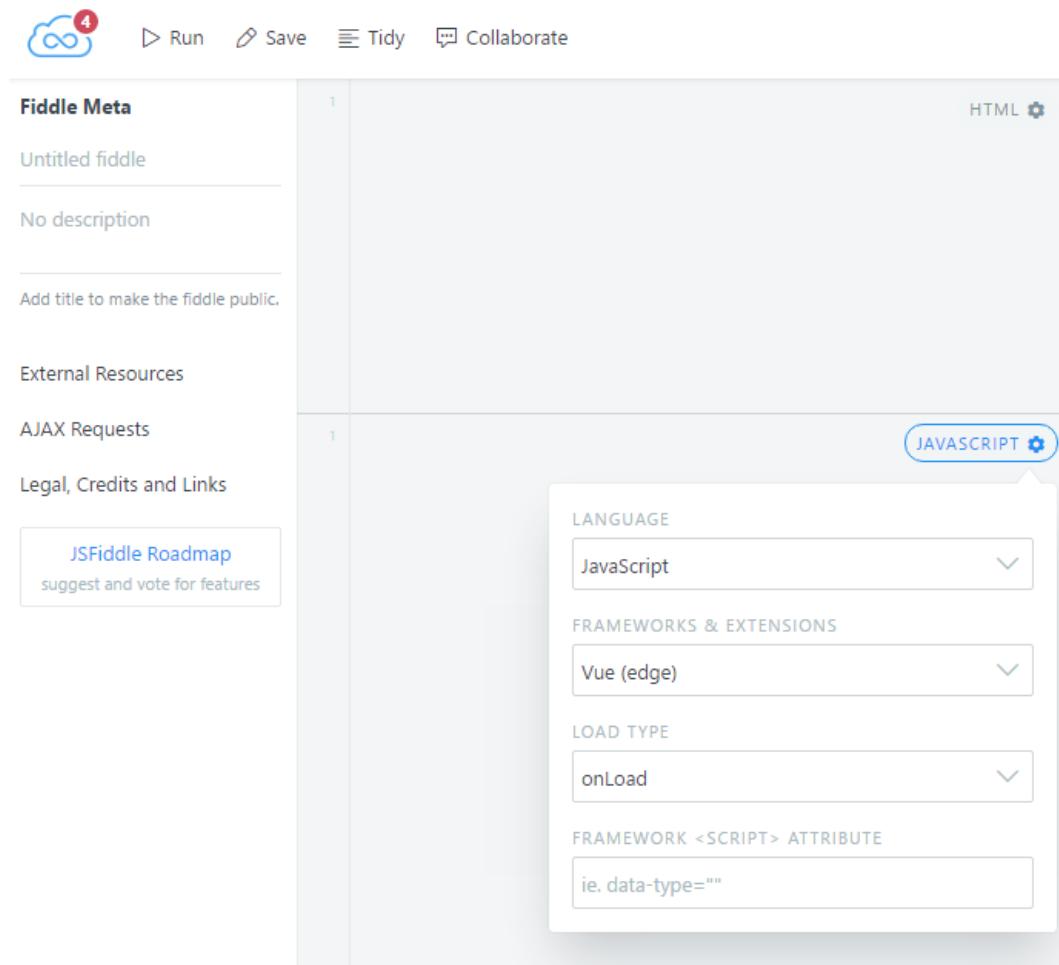
Caso queira, pode criar uma conta e salvar todos os códigos que criar, para poder consultar no futuro. No jsFiddle, temos 4 áreas sendo elas: *html*, *javascript*, *css* e *result*. Quando clicamos no botão Run, o *html/javascript/css* são combinados e o resultado é apresentado.

2.2 Configurando o jsFiddle para o Vue

Para que possamos usar o jsFiddle em conjunto com o vue, clique no ícone de configuração do javascript, de acordo com a figura a seguir:



Na caixa de seleção Frameworks & Extensions, encontre o item Vue e escolha a versão Vue (edge), deixando a configuração desta forma:



Agora que o jsFiddle está configurado com o Vue, podemos começar nosso estudo inicial com vue.

2.3 Hello World, vue

Para escrever um Hello World com vue apresentamos o conceito de databind. Isso significa que uma variável do vue será ligada diretamente a alguma variável no html.

Comece editando o código html, adicionando o seguinte texto:

```
1 <div id="app">
2   {{msg}}
3 </div>
```

Neste código temos dois detalhes. O primeiro, criamos uma div cujo o id é app. No segundo, usamos {{ e }} para adicionar uma variável chamada msg. Esta variável será preenchida pelo vue.

Agora vamos a parte Javascript. Para trabalhar com vue, basta criar um objeto Vue e repassar alguns parâmetros, veja:

```
1 new Vue({
2   el: '#app',
3   data: {
4     msg: 'Hello World'
5   }
6 })
```

O objeto criado new Vue() possui uma configuração no formato JSON, onde informamos a propriedade el, que significa o elemento em que esse objeto vue será aplicado no documento html. Com o valor #app, estamos apontando para a div cujo id é app.

A propriedade data é uma propriedade especial do Vue no qual são armazenadas todas as variáveis do objeto vue. Essas variáveis podem ser usadas tanto pelo próprio objeto vue quanto pelo databind. No nosso exemplo, a variável data.msg será ligada ao {{msg}} do html.

Após incluir estes dois códigos, clique no botão Run do jsFiddle. O resultado será semelhante a figura a seguir.

The screenshot shows a JSFiddle interface. On the left, there's a sidebar with links like 'Fiddle Author', 'Fiddle Meta', 'External Resources', 'AJAX Requests', 'Legal, Credits and Links', and a 'JSFiddle Roadmap' section with a 'suggest and vote for features' link. Below the roadmap is a 'Slack' advertisement. The main area has three tabs: 'HTML', 'CSS', and 'JAVASCRIPT'. The 'HTML' tab contains the code: <div id="app"> {{msg}} </div>. The 'JAVASCRIPT' tab contains: new Vue({ el: '.app', data: { msg: 'Hello World' } }). The output panel on the right shows the text 'Hello World'.

2.4 Two way databind

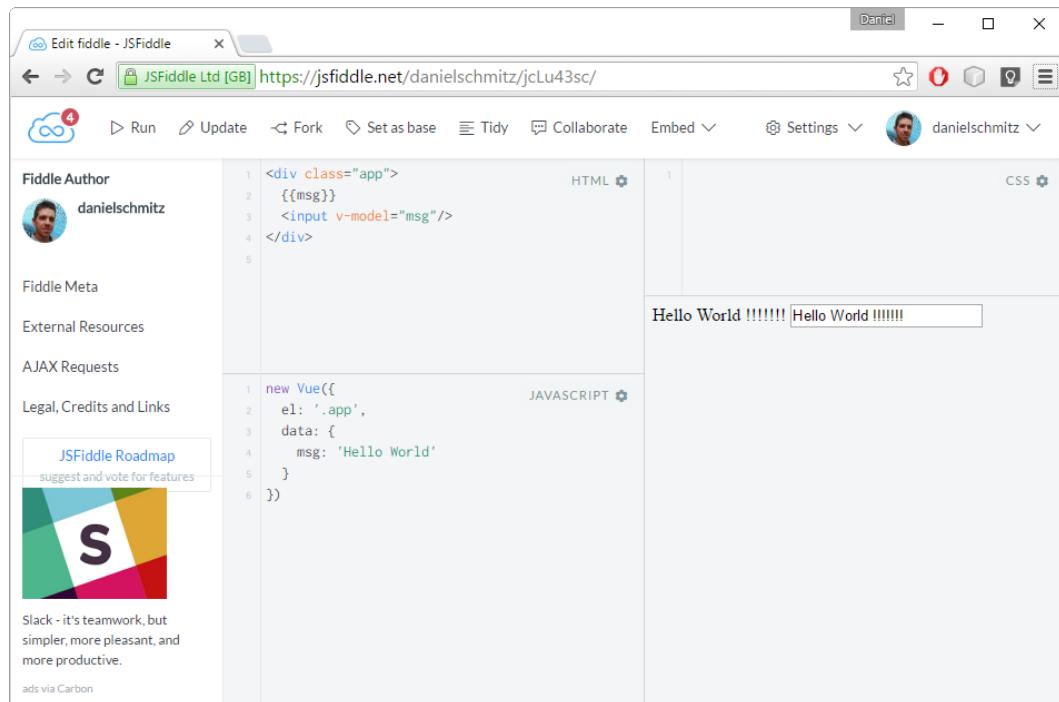
O conceito de “two-way” permite que o vue possa observar uma variável qualquer e atualizar o seu valor a qualquer momento. No exemplo a seguir, criamos um campo para alterar o valor da variável `msg`.

```

1 <div id="app">
2   {{msg}}
3   <input v-model="msg"/>
4 </div>

```

Veja que o elemento `input` possui a propriedade `v-model`, que é uma propriedade do vue. Ela permite que o vue observe o valor do campo `input` e atualize a variável `msg`. Quando alterarmos o valor do campo, a variável `data.msg` do objeto vue é alterada, e suas referências atualizadas. O resultado é semelhante à figura a seguir:



The screenshot shows a JSFiddle interface with the following details:

- HTML:** <div class="app"> {{msg}} </input v-model="msg"/> </div>
- JavaScript:** new Vue({ el: '.app', data: { msg: 'Hello World' } })
- Output:** Hello World !!!!!!!
- Left sidebar:** Fiddle Author (danielschmitz), Fiddle Meta, External Resources, AJAX Requests, Legal, Credits and Links, JSFiddle Roadmap (suggest and vote for features), Slack (it's teamwork, but simpler, more pleasant, and more productive), and ads via Carbon.

2.5 Criando uma lista

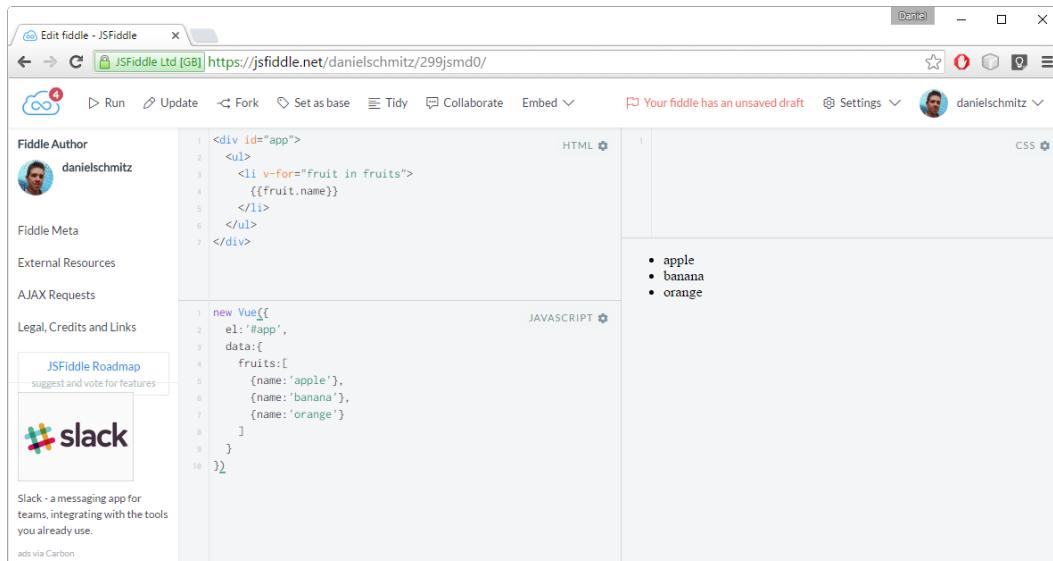
Existem dezenas de comandos do vue que iremos aprender ao longo desta obra. Um deles é o `v-for` que faz um loop no elemento em que foi inserido. Crie um novo jsFiddle com o seguinte código:

```
1 <div id="app">
2   <ul>
3     <li v-for="fruit in fruits">
4       {{fruit.name}}
5     </li>
6   </ul>
7 </div>
```

Veja que usamos `v-for` no elemento ``, que irá se repetir de acordo com a variável `fruits` declarada no objeto `vue`:

```
1 new Vue({
2     el: '#app',
3     data: {
4         fruits: [
5             {name: 'apple'},
6             {name: 'banana'},
7             {name: 'orange'}
8         ]
9     }
10 })
```

No objeto `vue`, cria-se o array `fruits` na propriedade `data`. O resultado é exibido a seguir:



The screenshot shows a JSFiddle interface with the following details:

- HTML:**

```
<div id="app">
  <ul>
    <li v-for="fruit in fruits">
      {{fruit.name}}
    </li>
  </ul>
</div>
```
- JAVASCRIPT:**

```
new Vue({
  el: '#app',
  data: {
    fruits:[
      {name:'apple'},
      {name:'banana'},
      {name:'orange'}
    ]
  }
})
```
- Output:** A list of fruits:
 - apple
 - banana
 - orange
- Left Sidebar:**
 - Fiddle Author: danielschmitz
 - Fiddle Meta
 - External Resources
 - AJAX Requests
 - Legal, Credits and Links
 - JSFiddle Roadmap
 - slack

2.6 Detectando alterações no Array

Vue consegue observar as alterações nos Arrays, fazendo com que as alterações no html sejam refletidas. O métodos que o Vue consegue observar são chamados de métodos modificadores, listados a seguir:

push()

Adiciona um item ao Array

pop()

Remove o último elemento do Array

shift()

Remove o primeiro elemento do Array

unshift()

Adiciona novos itens no início de um Array

splice()

Adiciona itens no Array, onde é possível informar o índice dos novos itens.

sort()

Ordena um Array

reverse()

Inverte os itens de um Array

Existem métodos que o Vue não consegue observar, chamados de *não modificadores*, tais como `filter()`, `concat()` e `slice()`. Estes métodos não provocam alterações no array original, mas retornam um novo Array que, para o Vue, poderiam ser sobrepostos inteiramente.

Por exemplo, ao usarmos `filter`, um novo array será retornado de acordo com a expressão de filtro que for usada. Quando subistituimos esse novo array, pode-se imaginar que o Vue irá remover o array antigo e recriar toda a lista novamente, mas ele não faz isso! Felizmente ele consegue detectar as alterações nos novos elementos do array e apenas atualizar as suas referências. Com isso substituir uma lista inteira de elementos nem sempre ocasiona em uma substituição completa na DOM.

Utilizando v-bind:key

Imagine que temos uma tabela com diversos registros sendo exibidos na página. Esses registros são originados em uma consulta Ajax ao servidor. Suponha que exista um filtro que irá realizar uma nova consulta, retornando novamente novos dados do servidor, que obviamente irão atualizar toda a lista de registros da tabela.

Perceba que, a cada pesquisa, uma nova lista é gerada e atualizada na tabela, o que pode se tornar uma operação mais complexa para a DOM, resultado até mesmo na substituição de todos os itens, caso o `v-for` não consiga compreender que os itens alterados são os mesmos.

Podemos ajudar o Vue nesse caso através da propriedade `key`, na qual iremos informar ao Vue qual propriedade da lista de objetos deverá ser mapeada para que o Vue possa manter uma relação entre os itens antes e após a reconsulta no servidor. Suponha, por exemplo, que a consulta no servidor retorne objetos que tenham uma propriedade chamada `_uid`, com os seguintes dados:

```
1  {
2    items: [
3      { _uid: '88f869d', ... },
4      { _uid: '7496c10', ... }
5    ]
6  }
```

Então pode-se dizer ao `v-for` que use essa propriedade como base da lista, da seguinte forma:

```
1  <div v-for="item in items" v-bind:key="_uid">
2
3  </div>
```

Desta forma, quando o Vue executar uma consulta ao servidor, e este retornar com novos dados, o `v-for` saberá pelo `uid` que alguns registros não se alteraram, e manterá a DOM intacta.

Uso do set

Devido a uma limitação do Javascript, o Vue não consegue perceber uma alteração no item de um array na seguinte forma:

```
1 this.fruits[0] = {}
```

Quando executamos o código acima, o item que pertence ao índice 0 do array, mesmo que alterado, não será atualizado na camada de visualização da página.

Para resolver este problema, temos que usar um método especial do Vue chamado `set`, da seguinte forma:

```
1 this.fruits.set(0, {name: 'foo'});
```

O uso do `set` irá forçar a atualização do Vue na lista html.

Como remover um item

Para que se possa remover um item, usamos o método `Array.prototype.splice` conforme o exemplo a seguir:

```
1 methods: {
2   removeTodo: function (todo) {
3     var index = this.todos.indexOf(todo)
4     this.todos.splice(index, 1)
5   }
6 }
```

Loops em objetos

Pode-se realizar um loop entre as propriedades de um objeto da seguinte forma:

```
1 <ul id="app" >
2   <li v-for="(value, key) in object">
3     {{ key }} : {{ value }}
4   </li>
5 </ul>
```

Onde `key` é o nome da propriedade do objeto, e `value` é o valor desta propriedade. Por exemplo, em um objeto `{"id": 5}`, o valor de `key` será `id` e o valor de `value` será `5`.

2.7 Eventos e métodos

Podemos capturar diversos tipos de eventos e realizar operações com cada um deles. No exemplo a seguir, incluímos um botão que irá alterar uma propriedade do vue.

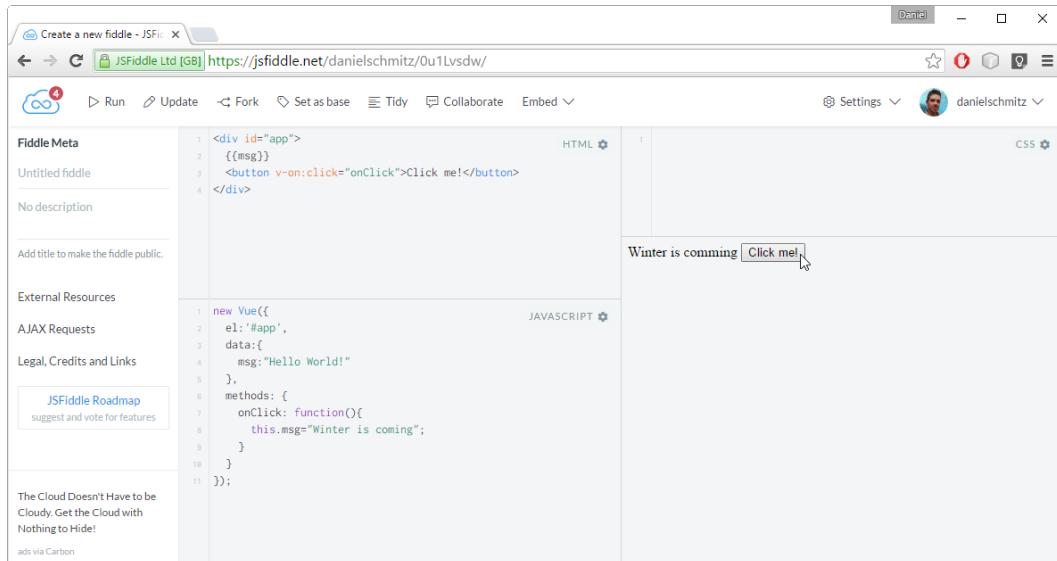
```
1 <div id="app" >
2   {{msg}}
3   <button v-on:click="onClick">Click me!</button>
4 </div>
```

No elemento `button` temos a captura do evento `click` pelo vue, representado por `v-on:click`. Esta captura irá executar o método `onClick`, que estará declarado no objeto Vue. O objeto é exibido a seguir:

```
1 new Vue({
2   el: '#app',
3   data: {
4     msg: "Hello World!"
5   },
6   methods: {
7     onClick: function(){
8       this.msg="Winter is coming";
9     }
10  }
11 })
```

```
9      }
10     }
11  ));
```

O objeto vue possui uma nova propriedade chamada `methods`, que reune todos os métodos que podem ser referenciados no código html. O método `onClick` possui em seu código a alteração da variável `msg`. O resultado deste código após clicar no botão é exibida a seguir:



```
<div id="app">
  {{msg}}
  <button v-on:click="onClick">Click me!</button>
</div>

new Vue({
  el:'#app',
  data:{
    msg:"Hello World!"
  },
  methods: {
    onClick: function(){
      this.msg="Winter is coming";
    }
  }
});
```

Modificando a propagação do evento

Quando clicamos em um botão ou link, o evento “click” irá executar o método indicado pelo `v-on:click` e, além disso, o evento se propagará até o navegador conseguir capturá-lo. Essa é a forma natural que o evento se comporta em um navegador.

Só que, às vezes, é necessário capturar o evento `click` mas não é desejável que ele se propague. Quando temos esta situação, é natural chamar o método `preventDefault` ou `stopPropagation`. O exemplo a seguir mostra como um evento pode ter a sua propagação cancelada.

```
1 <button v-on:click="say( 'hello!', $event)">
2   Submit
3 </button>
```

e:

```
1 methods: {
2   say: function (msg, event) {
3     event.preventDefault()
4     alert(msg)
5   }
6 }
```

O vue permite que possamos cancelar o evento ainda no html, sem a necessidade de alteração no código javascript, da seguinte forma:

```
1 <!-- a propagação do evento Click será cancelada -->
2 <a v-on:click.stop="doThis"></a>
3
4 <!-- o evento de submit não irá mais recarregar a página -->
5 <form v-on:submit.prevent="onSubmit"></form>
6
7 <!-- os modificadores podem ser encadeados -->
8 <a v-on:click.stop.prevent="doThat"></a>
```

Modificadores de teclas

Pode-se usar o seguinte modificador v-on:keyup.13 para capturar o evento “keyup 13” do teclado que corresponde a tecla enter. Também pode-se utilizar:

```
1 <input v-on:keyup.enter="submit">
2 ou
3 <input @keyup.enter="submit">
```

Algumas teclas que podem ser associadas:

- enter
- tab
- delete
- esc
- space
- up
- down
- left
- right

2.8 Design reativo

Um dos conceitos principais do Vue é o que chamamos de design reativo, onde elementos na página são alterados de acordo com o estado dos objetos gerenciados pelo Vue. Ou seja, não há a necessidade de navegar pela DOM (Document Object Model) dos elementos HTML para alterar informações.

2.9 Criando uma lista de tarefas

Com o pouco que vimos até o momento já podemos criar uma pequena lista de tarefas usando os três conceitos aprendidos até o momento:

- Pode-se armazenar variáveis no objeto Vue e usá-las no html
- Pode-se alterar o valor das variáveis através do two way databind
- Pode-se capturar eventos e chamar funções no objeto Vue

No código html, vamos criar um campo *input* para a entrada de uma tarefa, um botão para adicionar a tarefa e, finalmente, uma lista de tarefas criadas:

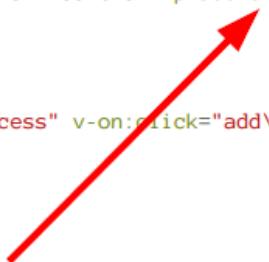
```
1 <div id="app" class="container">
2
3   <div class="row">
4     <div class="col-xs-8">
5       <input type="text" class="form-control" placeholder="\
6 Add a task" v-model="inputTask">
7     </div>
8     <div class="col-xs-4">
9       <button class="btn btn-success" v-on:click="addTask">
10      Adicionar
11      </button>
12    </div>
13  </div>
14  <br/>
15  <div class="row">
16    <div class="col-xs-10">
17      <table class="table">
18        <thead>
19          <tr>
20            <th>Task Name</th>
21            <th></th>
22          </tr>
23        </thead>
24        <tbody>
25          <tr v-for="task in tasks">
26            <td class="col-xs-11">
27              {{task.name}}
28            </td>
29            <td class="col-xs-1">
30              <button class="btn btn-danger" v-on:click="re\
31 moveTask(task.id)">
32                x
33              </button>
```

```
34          </td>
35      </tr>
36  </tbody>
37 </table>
38 </div>
39 </div>
40 </div>
```

Quebra de linha no código fonte

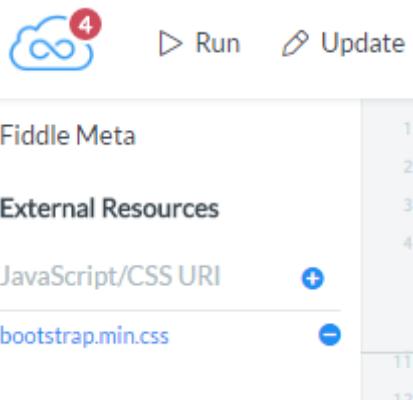
Cuidado com a quebra de página nos códigos desta obra. Como podemos ver na imagem a seguir:

```
<div class="row">
  <div class="col-xs-8">
    <input type="text" class="form-control" placeholder="Add a task" v-model="inputTask">
  </div>
  <div class="col-xs-4">
    <button class="btn btn-success" v-on:click="addTask">
      Adicionar
    </button>
  </div>
</div>
```



Quando uma linha quebra por não caber na página, é adicionado uma contra barra, que deve ser omitida caso você esteja copiando o código diretamente do arquivo PDF/EPUB desta obra.

Neste código HTML podemos observar o uso de classes nos elementos da página. Por exemplo, a primeira `<div>` contém a classe `container`. O elemento `input` contém a classe `form-control`. Essas classes são responsáveis em estilizar a página, e precisam de algum framework css funcionando em conjunto. Neste exemplo, usamos o Bootstrap, que pode ser adicionado no jsFiddle de acordo com o detalhe a seguir:



O endereço do arquivo adicionado é:

<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css>

Com o bootstrap adicionado, estilizar a aplicação torna-se uma tarefa muito fácil. Voltando ao html, criamos uma caixa de texto que possui como `v-model` o valor `inputTask`. Depois, adicionamos um botão que irá chamar o método `addTask`. A lista de tarefas é formada pelo elemento `table` e usamos `v-for` para criar um loop nos itens do array `tasks`.

O loop preenche as linhas da tabela, onde repassamos a propriedade `name` e usamos a propriedade `id` para criar um botão para remover a tarefa. Perceba que o botão que remove a tarefa tem o evento `click` associado ao método `removeTask(id)` onde é repassado o `id` da tarefa.

Com o html pronto, já podemos estabelecer que o objeto Vue terá duas variáveis, `inputTask` e `tasks`, além de dois métodos `addTask` e `removeTask`. Vamos ao código javascript:

```
1 new Vue({
2   el: '#app',
3   data: {
4     tasks: [
5       {id:1, name: "Learn Vue"},
6       {id:2, name: "Learn Npm"},
7       {id:3, name: "Learn Sass"}
8     ],
9     inputTask: ""
10  },
11  methods: {
12    addTask(){
13      if (this.inputTask.trim()!=""){
14        this.tasks.push(
15          {name:this.inputTask,
16           id:this.tasks.length+1}
17        )
18        this.inputTask="";
19      }
20    },
21    removeTask(id){
22      for(var i = this.tasks.length; i--;) {
23        if(this.tasks[i].id === id) {
24          this.tasks.splice(i, 1);
25        }
26      }
27    }
28  }
29})
```

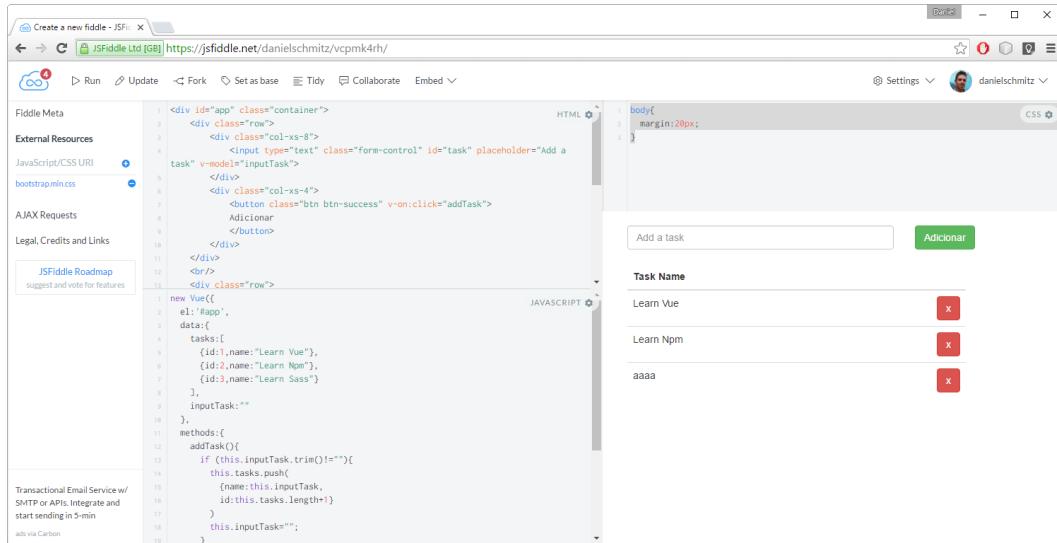
O código Vue, apesar de extenso, é fácil de entender. Primeiro criamos as duas variáveis: tasks possui um array de objetos que será a base da tabela que foi criada no html. Já a variável inputTask realiza um two way databind com a caixa de texto

do formulário, onde será inserida a nova tarefa.

O método `addTask()` irá adicionar uma nova tarefa a lista de tarefas `this.tasks`. Para adicionar esse novo item, usamos na propriedade `id` a quantidade de itens existentes da lista de tarefas.

O método `removeTask(id)` possui um parâmetro que foi repassado pelo botão html, e é através deste parâmetro que removemos a tarefa da lista de tarefas.

O resultado deste código pode ser visto na figura a seguir. Perceba que o formulário e a lista de tarefas possui o estilo do bootstrap.



The screenshot shows the JSFiddle interface with the following details:

- HTML:**

```
<div id="app" class="container">
  <div class="row">
    <input type="text" class="form-control" id="task" placeholder="Add a task" v-model="inputTask">
    <div class="col-xs-4">
      <button class="btn btn-success" v-on:click="addTask">Adicionar</button>
    </div>
  </div>
</div>
```
- CSS:**

```
body{
  margin:20px;
```
- JAVASCRIPT:**

```
new Vue({
  el: '#app',
  data: {
    tasks: [
      {id:1, name: "Learn Vue"},
      {id:2, name: "Learn Npm"},
      {id:3, name: "Learn Sass"}
    ],
    inputTask: ''
  },
  methods: {
    addTask(){
      if (this.inputTask.trim()!=""){
        this.tasks.push(
          {name:this.inputTask,
          id:this.tasks.length+1}
        )
        this.inputTask="";
      }
    }
  }
})
```
- Rendered Output:**

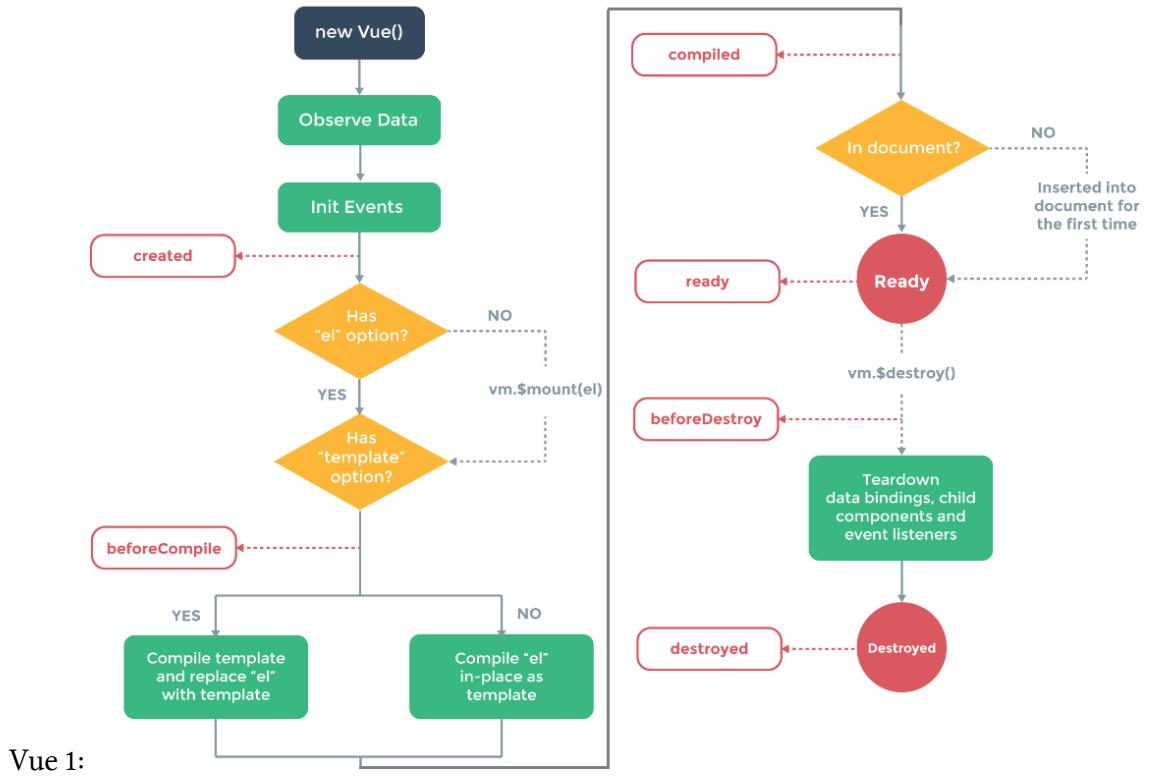
Add a task

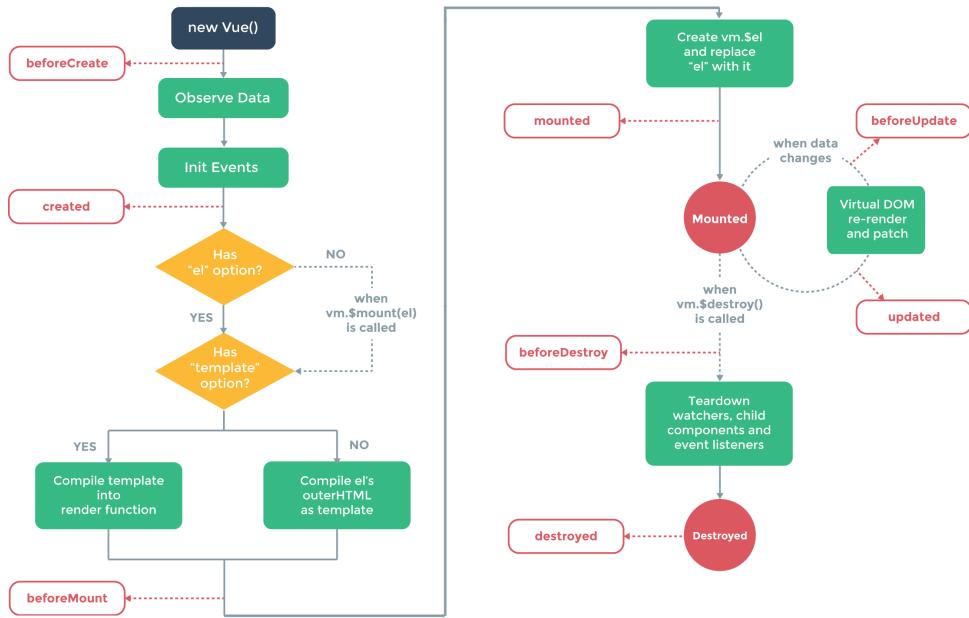
Task Name
Learn Vue
Learn Npm
aaaa

Caso queira ter acesso direto a este código pelo jsFiddle, (clique aqui) [<https://jsfiddle.net/tbg8fo51/>]

2.10 Eventos do ciclo de vida do Vue

Quando um objeto Vue é instanciado, ele possui um ciclo de vida completo, desde a sua criação até a finalização. Este ciclo é descrito pela imagem a seguir:





Vue 2:

Em vermelho, temos os eventos que são disparados durante este ciclo, e que podem ser usados em determinados ocasiões no desenvolvimento de sistemas. Os eventos que podem ser capturados são: beforeCreated, created, beforeMount, mounted, beforeUpdate, updated, beforeDestroy, destroyed.

Para realizar requisições ajax, costuma-se utilizar o evento updated. Veremos com mais detalhes este tópico quando abordarmos vue-resource.

2.11 Compreendendo melhor o Data Bind

Existem alguns pequenos detalhes que precisamos abordar sobre o databind. Já sabemos que, ao usar `{} {}` conseguimos referenciar uma variável do Vue diretamente no html.

Databind único

Caso haja a necessidade de aplicar o data-bind somente na primeira vez que o Vue for iniciado, deve-se usar `v-once`, conforme o código a seguir:

```
1 <span v-once> Message: {{ msg }} </span>
```

Databind com html

O uso de `{} e {}` não formata o html por uma simples questão de segurança. Isso significa que, se você tiver na variável `msg` o valor `Hello World`, ao usar `{}{{msg}}{}` a resposta ao navegador será `'Hello World'`, de forma que as tags do html serão exibidas, mas não formatadas.

Para que você possa formatar código html no databind, é necessário usar três a diretiva `v-html`, como por exemplo `{{msg}}`. Desta forma, o valor `Hello World` será exibido no navegador em negrito.

Databind em Atributos

Para usar data-bind em atributos, use a directive `v-bind`, como no exemplo a seguir:

```
1 <button v-bind:class="'btn btn-' + size></button>
```

Expressões

Pode-se usar expressões dentro do databind, como nos exemplos a seguir:

`{}{{ number + 1 }}{}`

Se `number` for um número e estiver declarado no objeto Vue, será adicionado o valor 1 à ele.

`{}{{ ok ? 'YES' : 'NO' }}{}`

Se `ok` for uma variável booleana, e se for verdadeiro, o texto YES será retornado. Caso contrário, o texto NO será retornado.

`{}{{ message.split('').reverse().join('') }}{}`

Nesta expressão o conteúdo da variável `message` será quebrada em um array, onde cada letra será um item deste array. O método `reverse()` irá reverter os indices do array e o método `join` irá concatenar os itens do array em uma string. O resultado final é uma string com as suas letras invertidas.

É preciso ter alguns cuidados em termos que são sentenças e não expressões, como nos exemplos `{} var a = 1 {}` ou então `{} if (ok) {return message} {}`. neste caso não irão funcionar

2.12 Filtros

Filtros são usados, na maioria das vezes, para formatar valores de databind. O exemplo a seguir irá pegar todo o valor de `msg` e transformar a primeira letra para maiúscula.

```
1 <div>
2   {{ msg | capitalize }}
3 </div>
```

Na versão 2.0 do Vue, os filtros somente funcionam em expressões `{} ... {}`. Filtros em laços `v-for` não são mais usados.

O nome do filtro a ser aplicado deve estar após o caractere `pipe |`. `Capitalize` é somente um dos filtros disponíveis. Existem alguns pré configurados, veja:

2.13 Diretivas

As diretivas do Vue são propriedades especiais dos elementos do html, geralmente se iniciam pela expressão `v-` e possui as mais diversas funcionalidades. No exemplo a seguir, temos a diretiva `v-if` em ação:

```
1 <p v-if="greeting">Hello!</p>
```

O elemento `<p>` estará visível ao usuário somente se a propriedade `greeting` do objeto Vue estiver com o valor `true`.

Ao invés de exibir uma lista completa de diretivas (pode-se consultar a [api²](http://vuejs.org/api), sempre), vamos apresentar as diretivas mais usadas ao longo de toda esta obra.

Na versão do Vue 2, a criação de diretivas personalizadas não é encorajada, pois o Vue foca principalmente na criação de componentes.

²<http://vuejs.org/api>

Argumentos

Algumas diretivas possuem argumentos que são estabelecidos após o uso de dois pontos. Por exemplo, a diretiva `v-bind` é usada para atualizar um atributo de qualquer elemento html, veja:

```
1 <a v-bind:href="url"></a>
2
3 é o mesmo que:
4
5 <a :href="url"></a>
6
7 ou então:
8
9 <a href="{{url}}></a>
```

Outro exemplo comum é na diretiva `v-on`, usada para registrar eventos. O evento `click` pode ser registrado através do `v-on:click`, assim como a tecla `enter` pode ser associada por `v-on:keyup.enter`.

Modificadores

Um modificador é um elemento que ou configura o argumento da diretiva. Vimos o uso do modificador no exemplo anterior, onde `v-on:keyup.enter` possui o modificador “`enter`”. Todo modificador é configurado pelo ponto, após o uso do argumento.

2.14 Atalhos de diretiva (Shorthands)

Existem alguns atalhos muito usados no Vue que simplificam o código html. Por exemplo, ao invés de termos:

```
1 <input v-bind:type="form.type"  
2 v-bind:placeholder="form.placeholder"  
3 v-bind:size="form.size">
```

Podemos usar o atalho:

```
1 <input :type="form.type"  
2 :placeholder="form.placeholder"  
3 :size="form.size">
```

Ou seja, removemos o “v-bind:” e deixamos apenas p “:”. Sempre quando ver um “:” nos elementos que o Vue gerencia, lembre-se do v-bind.

Outro atalho muito comum é na manipulação de eventos, onde trocamos o v-on: por @. Veja o exemplo:

```
1 <!-- full syntax -->  
2 <a v-on:click="doSomething"></a>  
3  
4 <!-- shorthand -->  
5 <a @click="doSomething"></a>
```

2.15 Alternando estilos

Uma das funcionalidades do design reativo é possibilitar a alteração de estilos no código html dado algum estado de variável ou situação específica.

No exemplo da lista de tarefas, no método addTask, tínhamos a seguinte verificação no código javascript:

```
1 addTask(){
2   if (this.inputTask.trim()!=""){
3     //.....
4   }
5 }
```

Com design reativo podemos alterar, por exemplo, o botão que inclui a tarefa para:

```
1 <button class="btn btn-success"
2 :class="{ 'disabled':inputTask.trim()==' ' }"
3 @click="addTask"
4 >
```

Teste esta variação neste [link³](#)

Veja que adicionamos o *shorthand* `:class` incluindo a classe `disabled` do bootstrap, fazendo a mesma verificação para que, quando não houver texto digitado na caixa de texto, o botão `Adicionar` ficará desabilitado, conforme a figura a seguir.

³<https://jsfiddle.net/0wbqvw3v/1/>

The screenshot shows a JSFiddle interface with the following details:

- HTML:**

```
<div id="app" class="container">
  <div class="row">
    <div class="col-xs-8">
      <input type="text" class="form-control" id="task" placeholder="Add a task" v-model="inputTask">
    </div>
  </div>
</div>
```
- CSS:**

```
body{
  margin:20px;
}
```
- JavaScript:**

```
new Vue({
  el:'#app',
  data:{
    tasks:[
      {id:1,name:'Learn Vue'},
      {id:2,name:'Learn Npm'},
      {id:3,name:'Learn Sass'}
    ],
    inputTask:""
  },
  methods:{
    addTask(){
      if (this.inputTask.trim()!=""){
        this.tasks.push(
          {name:this.inputTask,
          id:this.tasks.length+1}
        )
        this.inputTask="";
      }
    }
  }
})
```
- Preview:**
 - Form with input field "Add a task" and a green "Adicionar" button.
 - Table with three rows:

Task Name	
Learn Vue	X
Learn Npm	X
Learn Sass	X

2.16 Uso da condicional v-if

O uso do `v-if` irá exibir o elemento html de acordo com alguma condição. Por exemplo:

```
1 <h1 v-if="showHelloWorld">Hello World</h1>
```

É possível adicionar `v-else` logo após o `v-if`, conforme o exemplo a seguir:

```
1 <h1 v-if="name==' '">Hello World</h1>
2 <h1 v-else>Hello {{name}}</h1>
```

O `v-else` tem que estar imediatamente após o `v-if` para que possa funcionar.

A diretiva `v-if` irá incluir ou excluir o item da DOM do html. Caso haja necessidade de apenas omitir o elemento (usando `display:none`), usa-se `v-show`.

2.17 Exibindo ou ocultando um bloco de código

Caso haja a necessidade de exibir um bloco de código, pode-se inserir `v-if` em algum elemento html que contém esse bloco. Se não houver nenhum elemento html englobando o html que se deseja tratar, pode-se usar o componente `<template>`, por exemplo:

```
1 <template v-if="ok">
2   <h1>Title</h1>
3   <p>Paragraph 1</p>
4   <p>Paragraph 2</p>
5 </template>
```

2.18 v-if vs v-show

A diferença principal entre `v-if` e `v-show` está na manipulação do DOM do html. Quando usa-se `v-if`, elementos são removidos ou inseridos na DOM, além de todos os data-binds e eventos serem removidos ou adicionados também. Isso gera um custo de processamento que pode ser necessário em determinadas situações.

Já o `v-show` usa estilos apenas para esconder o elemento da página, mas não da DOM, o que possui um custo baixo, mas pode não ser recomendado em algumas situações.

2.19 Formulários

O uso de formulários é amplamente requisitado em sistemas ambientes web. Quanto melhor o domínio sobre eles mais rápido e eficiente um formulário poderá ser criado.

Para ligar um campo de formulário a uma variável do Vue usamos `v-model`, da seguinte forma:

```
1 <span>Message is: {{ message }}</span>
2 <br>
3 <input type="text" v-model="message">
```

Checkbox

Um input do tipo checkbox deve estar ligado a uma propriedade do v-model que seja booleana. Se um mesmo v-model estiver ligado a vários checkboxes, um array com os itens selecionados será criado.

Exemplo:

```
1 <input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
2 <label for="jack">Jack</label>
3 <input type="checkbox" id="john" value="John" v-model="checkedNames">
4 <label for="john">John</label>
5 <input type="checkbox" id="mike" value="Mike" v-model="checkedNames">
6 <label for="mike">Mike</label>
7 <br>
8 <span>Checked names: {{ checkedNames | json }}</span>
```

```
1 new Vue({
2   el: '#app',
3   data: {
4     checkedNames: []
5   }
6 })
```

Resultado:



Radio

Campos do tipo Radio só aceitam um valor. Para criá-los, basta definir o mesmo v-model e o Vue irá realizar o databind.

Select

Campos do tipo select podem ser ligados a um v-model, onde o valor selecionado irá atualizar o valor da variável. Caso use a opção multiple, vários valores podem ser selecionados.

Para criar opções dinamicamente, basta usar v-for no elemento <option> do select, conforme o exemplo a seguir:

```
1 <select v-model="selected">
2   <option v-for="option in options" v-bind:value="option.value">
3     {{ option.text }}
4   </option>
5 </select>
```

Atributos para input

Existem três atributos que podem ser usados no elemento input para adicionar algumas funcionalidades extras. São eles:

lazy Atualiza o model após o evento change do campo input, que ocorre geralmente quando o campo perde o foco. Exemplo: `<input v-model.lazy="name">`

number

Formata o campo input para aceitar somente números.

2.20 Conclusão

Abordamos quase todas as funcionalidades do vue e deixamos uma das principais, *Components*, para o próximo capítulo, que necessita de uma atenção em especial.