



UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Fundamentos de Vue.js

QXD0193 - Programação de Interfaces Web

Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Agenda

- Introdução
- Introdução ao VueJs
- Principais aspectos de uma aplicação em VueJs
- Diretivas
- Componentes
- Prática

Introdução



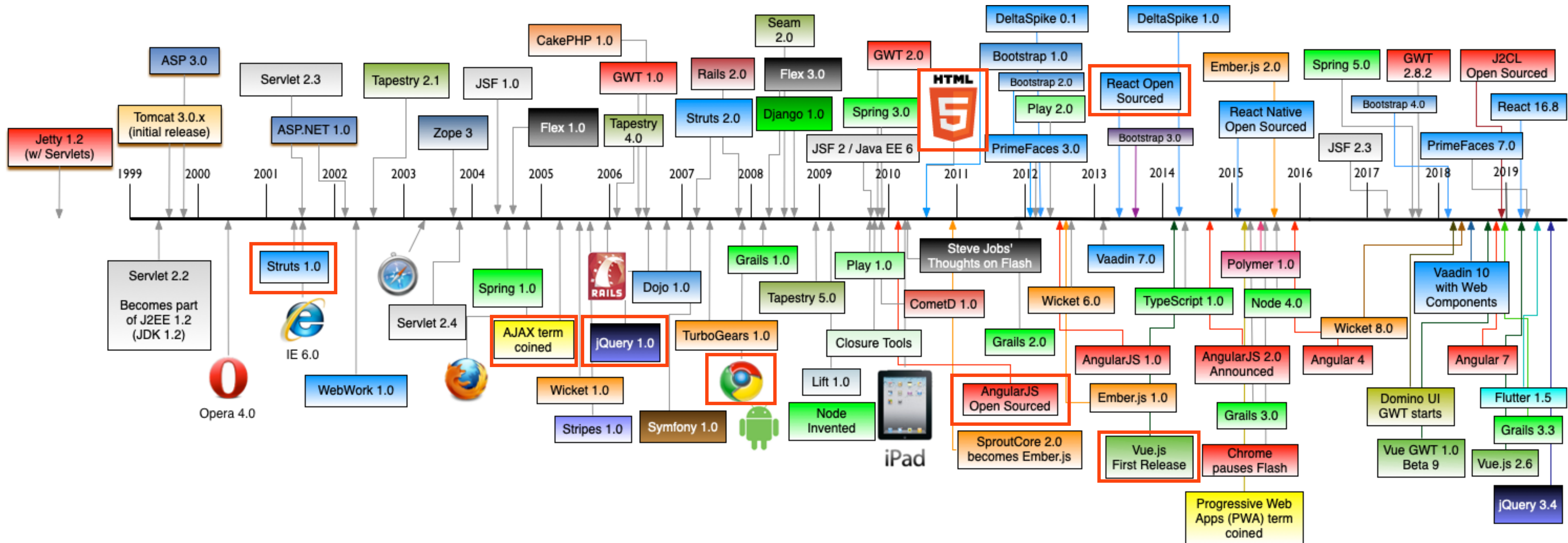
Introdução

Motivação

- A interatividade em websites sempre atraiu a atenção de desenvolvedores
- Com o desenvolvimento da Web 2.0, nos anos 2000, a interatividade e o engajamento do usuário passaram a receber um foco ainda maior
 - Companhias como Twitter, Facebook, and YouTube foram criadas nesse período
- Desenvolvedores precisaram se adaptar para permitir esse novo nível de interatividade
 - Bibliotecas e framework foram lançadas para permitir a criação desses sites
 - Em 2006, John Resig lançou o jQuery, que simplificou a escrita de JS no lado do cliente
 - Com o passar do tempo, bibliotecas e frameworks focando no server side também surgiram

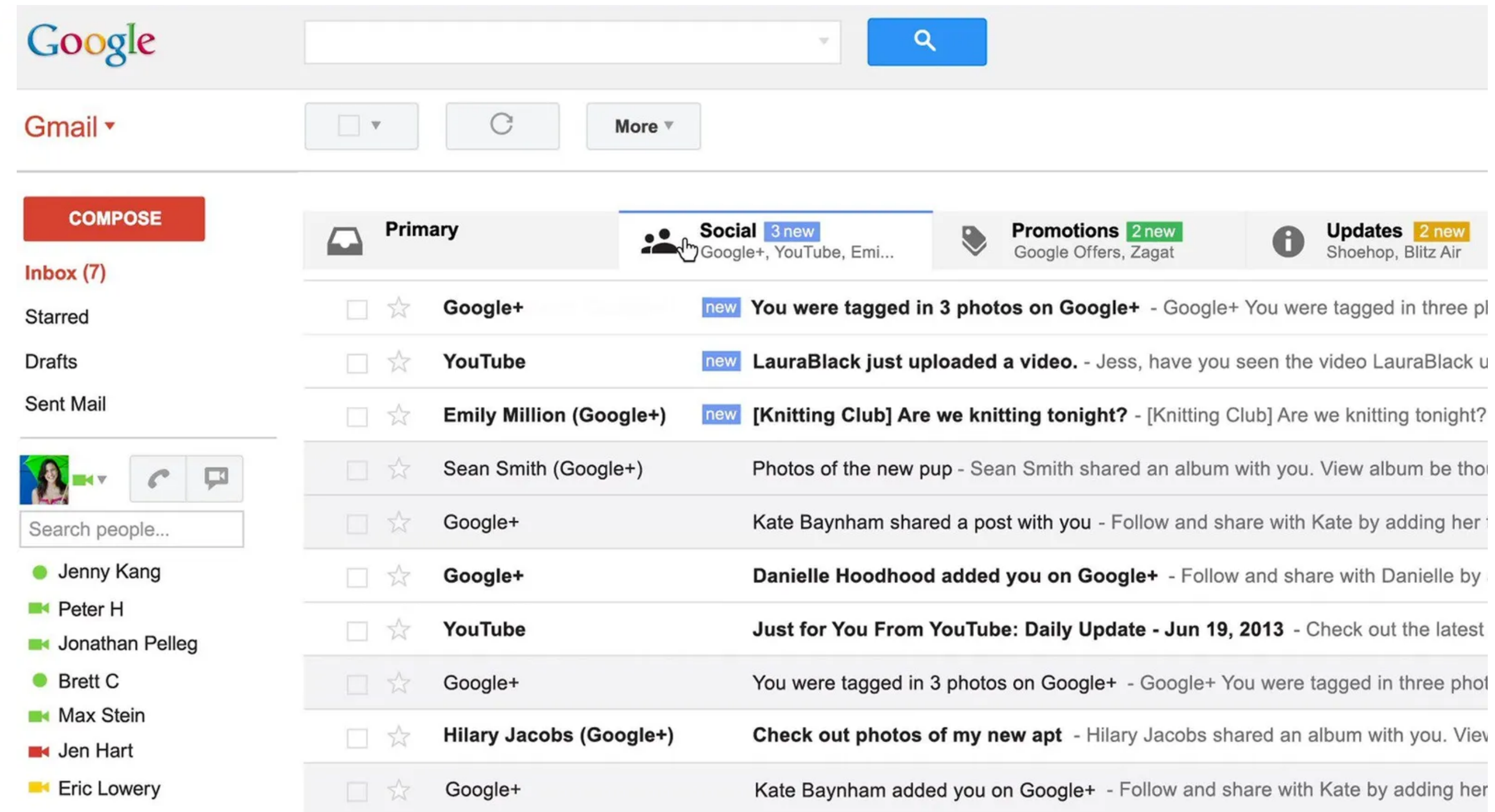
Introdução

Motivação



Introdução

Motivação



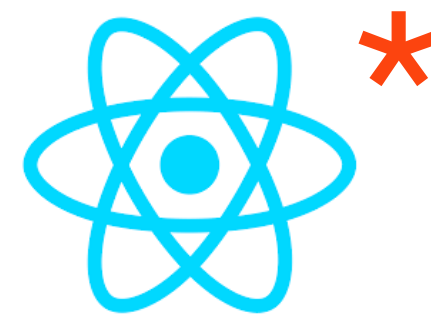
Introdução

MVC -> MVVM

- O suporte ao AJAX (2005) permitiu a atualização parcial de aplicações web
 - Requisições a página completa deixaram de ser necessárias
 - As atualizações passaram a ser mais rápidas
 - No entanto, algum esforço duplicado era necessário com o espelhamento da lógica de apresentação e a lógica de negócio
- Por volta de 2010 os primeiros framework focados em MVVM surgiram



- Atualmente os frameworks mais utilizados utilizam a arquitetura MVVM



Introdução

MVC -> MVVM

- MVVM (Model View ViewModel) é um padrão arquitetural baseado no MVC
 - Foca em separar mais claramente a UI da lógica de negócio da aplicação (*model*)
- Diversas implementações desse padrão utilizam *declarative data binding* para separar o a implementação das *Views* de outra camadas
- Foi projetado para "remover" todo o código relacionado UI por meio do uso de *data binding*
- A ideia é obter ambas as vantagens da separação do desenvolvimento funcional fornecido pelo MVC, enquanto aproveita as vantagens do *data binding*

Introdução

MVVM - View

- Responsável por mostrar as informações para o usuário
- Representam o estado atual da aplicação fornecido pelo **ViewModel**
- São ativas já que possuem data-bindings, eventos e comportamentos que requerem conhecimento do **Model** e do **ViewModel**
- São responsáveis por tratar os eventos do usuário e enviá-los ao **ViewModel**
- É importante notar que a **View** não é responsável por manter o estado da aplicação

Introdução

MVVM - ViewModel

- Pode ser considerado como um **Controller** especializado que age como um conversor de dados
 - Transforma as informações do **Model** para **View** passando comandos da View para o Model
- Armazena o estado atual da aplicação e gerencia boa parte da lógica de visualização
 - Mantém a sincronia do estado da aplicação entre com **View** por meio do **data binding (Binder)**

Introdução

MVVM - Model

- Como em outros padrões da família **MV***, o **Model** representa os dados específicos do domínio da aplicação
- Armazenam informações, mas não tratam o comportamento (lógica de negócio)
- É a camada responsável por persistir os dados da aplicação

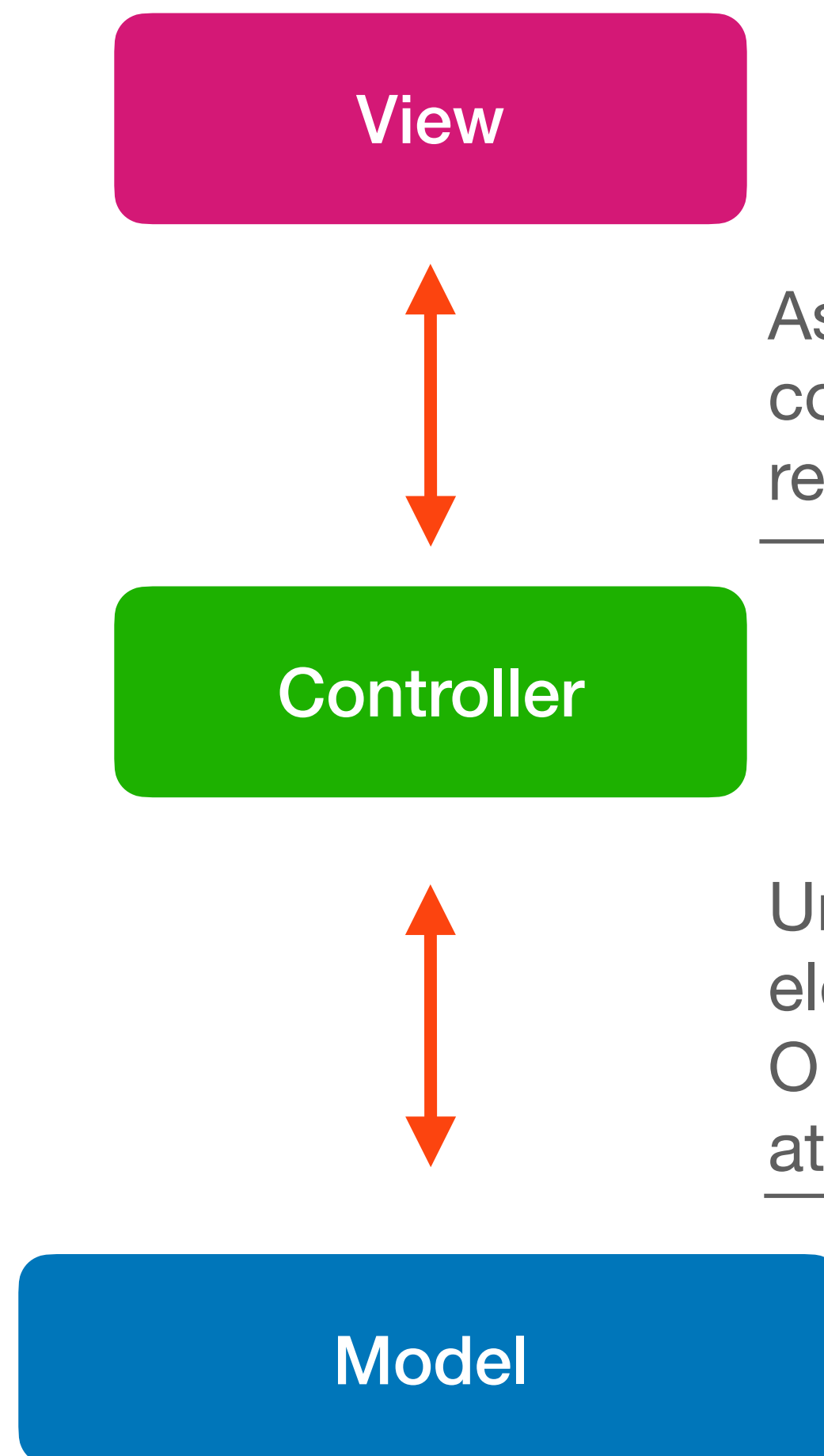
Introdução

MVC -> MVVM

Uma view é responsável por mostrar as informações para o usuário

O controller age como um mediator. Ele ajuda na aquisição e transformação de dados a partir do model e as propaga para a View. Além disso, ele roteia os eventos ocorridos na View até o Model.

Em uma aplicação MVC, o modelo é apresentado pelo domínio dela e sua regra de negócio.



As ações do usuário nas Views são enviado ao controller. Ao final do processo de responder a requisição, a resposta é enviada para um nova view.

Uma vez que o Controller processa a informação, ele as envia ao modelo para que sejam persistidas. O modelo informa ao Controller quando essa atividade termina.

Introdução

MVC -> MVVM

Apresenta o estado atual da aplicação fornecido pelo ViewModel e lidam com eventos do usuário

ViewModel não apenas expõe atributos dos modelos, mas também dão acesso a métodos e funcionalidades como validação. Armazena o estado atual da aplicação e gerencia boa parte da lógica de visualização.

Model armazenam informações, mas não tratam o comportamento (lógica de negócio).



Views e ViewModel se comunicam via data-binding e eventos. O binder expõe os dados para View em forma de propriedades. Models e propriedades no ViewModel são sincronizados e atualizados via two-way data-binding.

O ViewModel, assim com o Controller, é o responsável enviar as informações que devem ser persistidas pelo Model.

Introdução

MVVM - Vantagens

- Facilita o desenvolvimento em paralelo da UI e de seus components
- Abstrai o funcionamento da View, reduzindo a quantidade da lógica de negócio na camada de View
- O **ViewModel** é mais fácil de ser testado com testes unitários se comparado com *event-driven code*
- O **ViewModel** pode ser testado sem preocupações com automatização da UI e interações
- Da um melhor suporte a aplicações **reativas**

Introdução

Aplicações Reativas

- Não são um paradigma ou uma nova ideia
- Sua adoção no contexto web está intimamente ligada a framework JS como: Vue, React e Angular
- De forma simplificada podemos dizer que uma aplicação reativa:
 - **Observa** as modificações do estado da aplicação
 - **Propaga/Notifica** as mudanças em toda a aplicação
 - **Atualiza**/Renderiza as views automaticamente em resposta a mudanças
 - Forneça feedback oportuno para as interações do usuário

Introdução ao VueJS



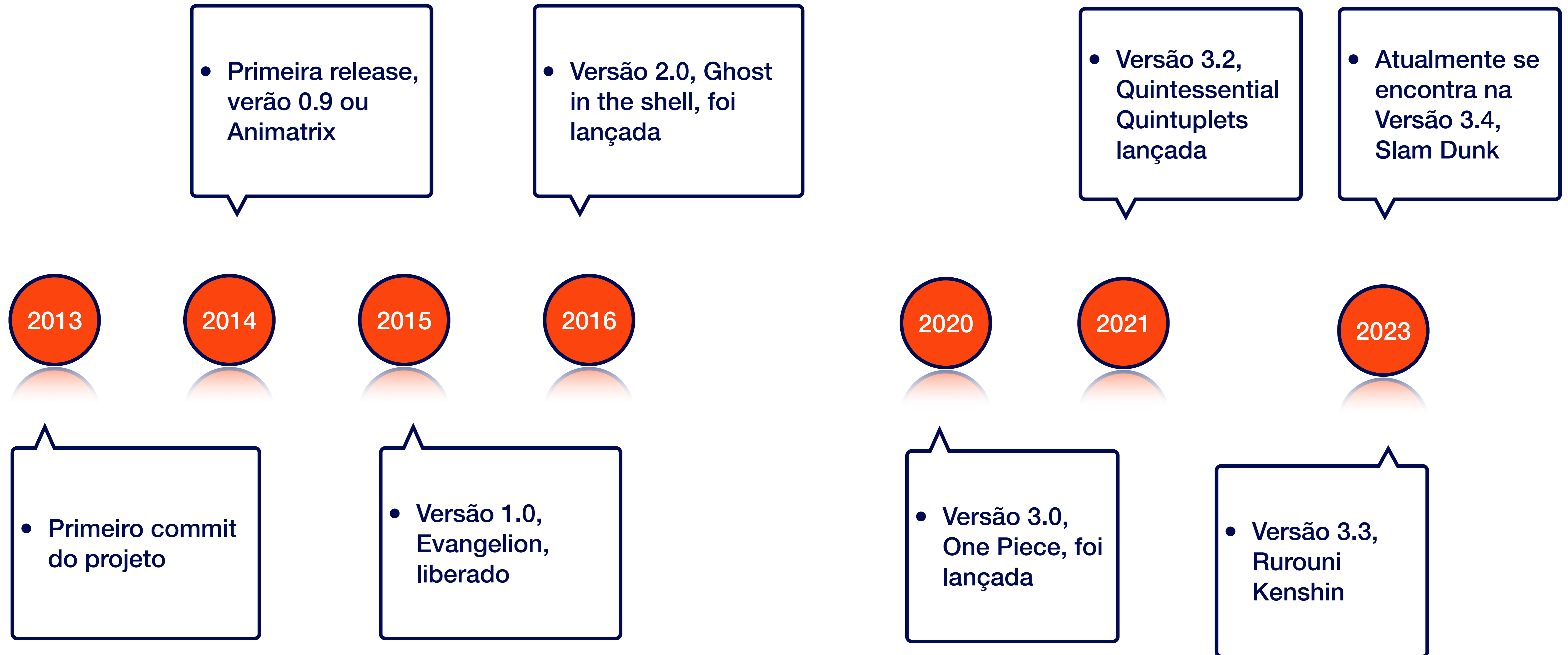
Introdução ao VueJS

VueJs

- Comumente conhecido como Vue, pronunciado “view”
- Framework **progressivo** do JavaScript de código aberto (*open source*) para a construção de interfaces de usuário
 - Projetado para ser adotado de forma incremental
- Também pode funcionar como uma estrutura de aplicativos web capaz de alimentar aplicativos avançados de uma única página
- Criado por **Evan You** depois de trabalhar para o **Google** no **AngularJS**

Introdução

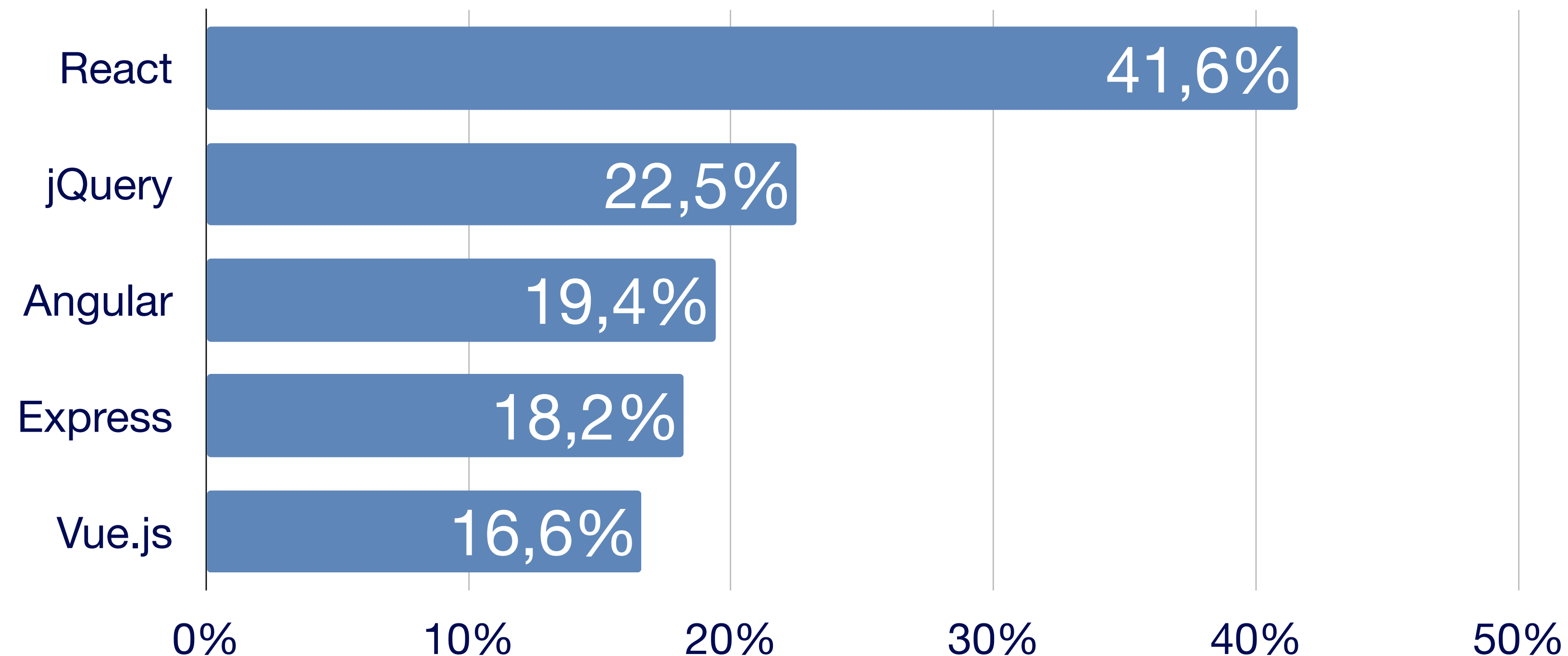
História - Timeline



Introdução ao VueJS

Comunidade

Frameworks web mais utilizados

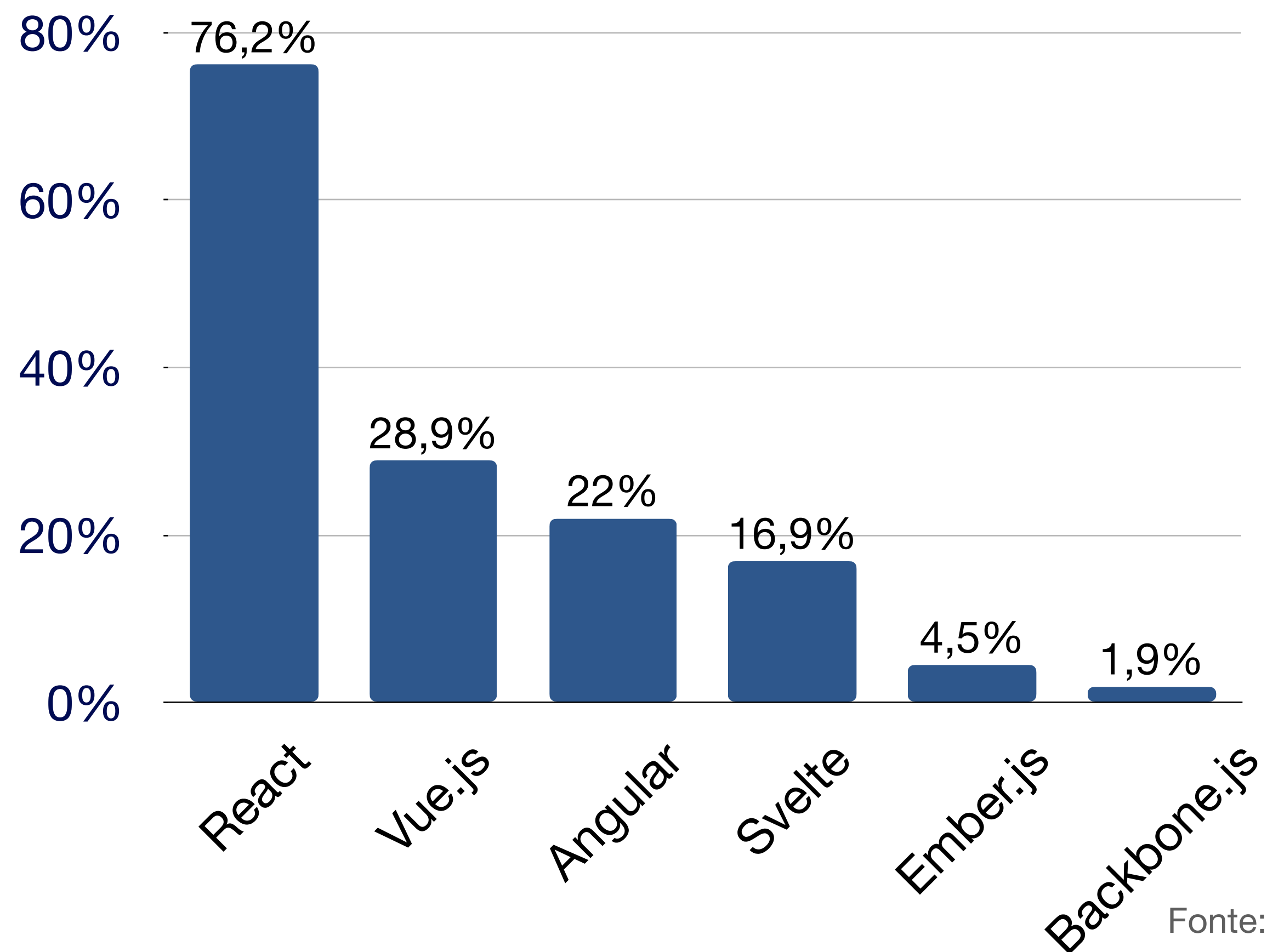


Fonte: <https://survey.stackoverflow.co/2024/technology/>

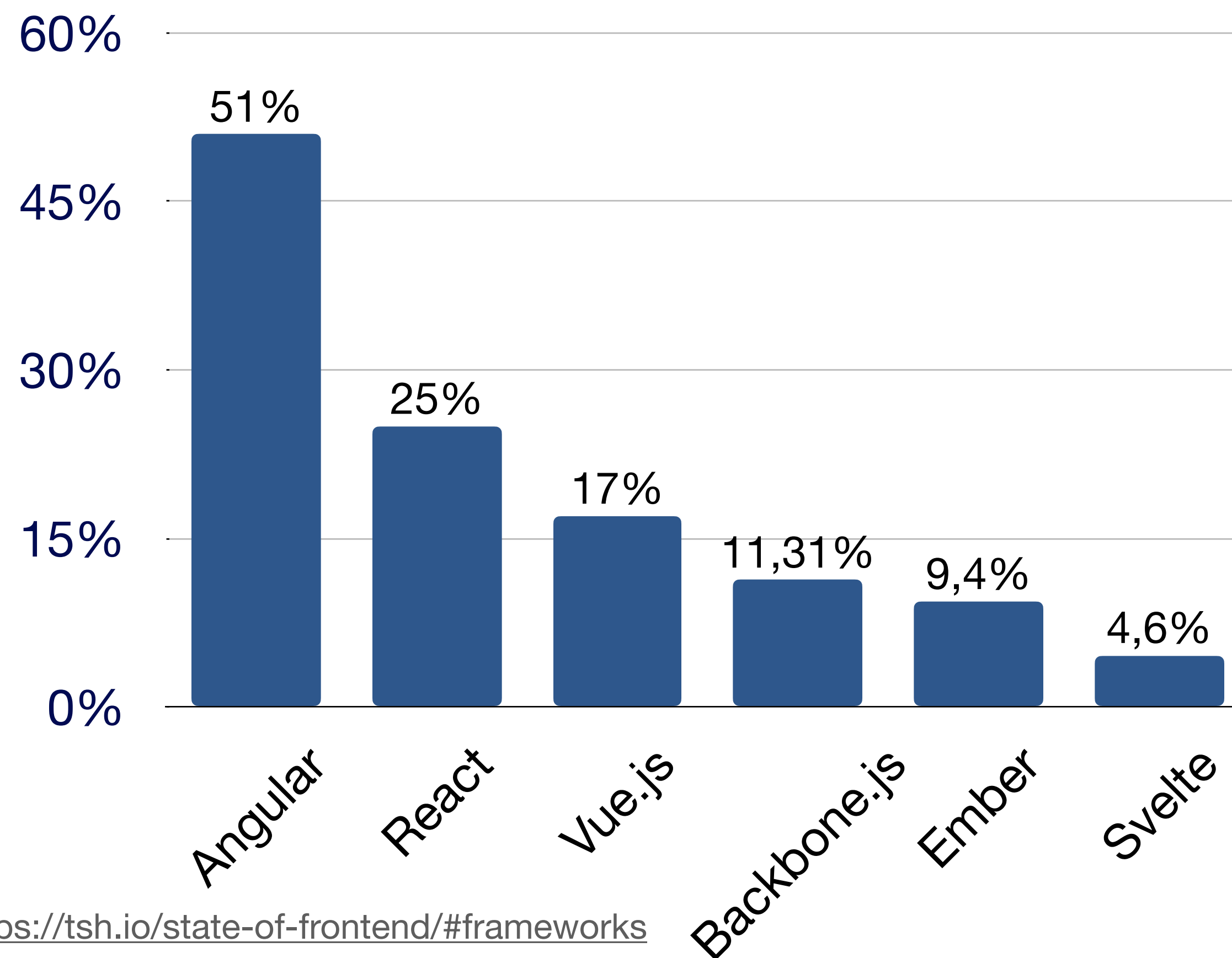
Introdução ao VueJS

Comunidade

Quais frameworks você utilizou no último ano e gostou?



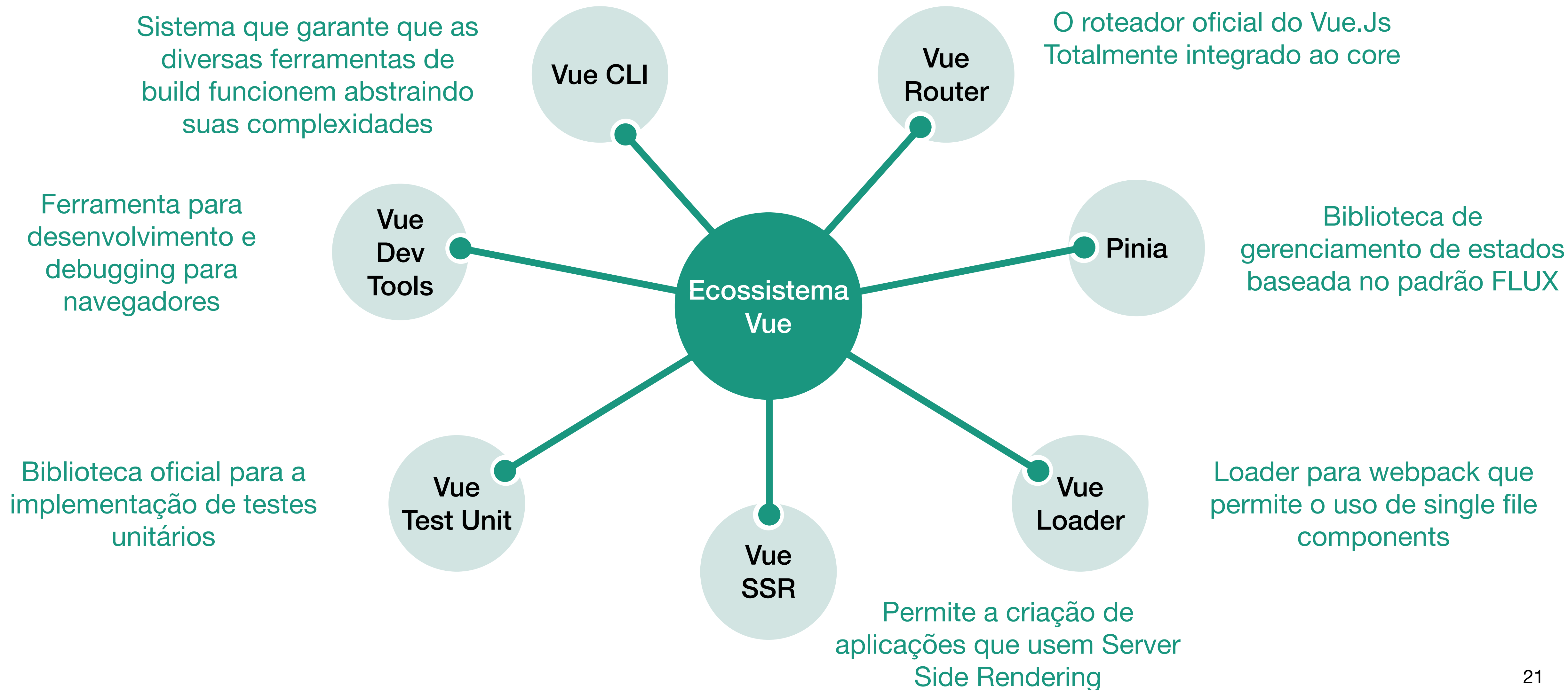
Qual framework você usou e não gostou?



Fonte: <https://tsh.io/state-of-frontend/#frameworks>

Introdução ao VueJS

Ecossistema



Introdução ao VueJS

Vantagens

- Leve
 - Pouco mais de 18KB, consideravelmente menor que os concorrentes
- Performance e Virtual DOM
- Data binding reativo de duas mãos (*two way*)
- Legibilidade
 - Single File Components incentivam a separação em componentes com seus respectivos HTML, CSS e JS/TS

Introdução ao VueJS

Vantagens

- Ecossistema bem estabelecido
- Flexível
- Documentação concisa e atualizada
- Suporte da comunidade
- Fácil de usar

Criando uma aplicação Vue



Criando uma aplicação Vue

```
const { createApp } = Vue
const app = createApp({
  /* options */
}).mount('#app')
```

O pontapé inicial de toda aplicação em Vue é a criação de uma instância do objeto *application*

Recebe como primeiro parâmetro um objeto usado para configurar o componente raiz (root)

Uma aplicação precisa ser montando em um elemento da DOM

Criando uma aplicação Vue

```
const { createApp } = Vue
const RootComponent = { /* options */ }
createApp(RootComponent)
  .component('SearchInput', SearchInputComponent)
  .directive('focus', FocusDirective)
  .use(LocalePlugin)
  .mount('#app')
```

O componente raiz é o ponto inicial de renderização após a montagem da aplicação

A instância de application é utilizada para registrar elementos “globais” que podem ser usados por outros componentes da aplicação

Criando uma aplicação Vue

```
const { ref, createApp } = Vue
const vm = createApp({
  setup() {
    const count = ref(4)
    return { count }
  }
}).mount('#app')
console.log(vm.count) // -> 4
```

- Cada componente pode expor por meio do retorno da função *setup*
 - Deve retornar um objeto
 - Automaticamente são rastreados pelo sistema de reatividade do Vue

Criando uma aplicação Vue

Data binding, interpolação e Diretivas

```
const vm = createApp({
  setup() {
    const count = ref(4)
    return { count }
  }
}).mount('#app')
vm.count++
```

```
<span>Message: {{ count }}</span>
```

Message: 4

- A forma mais simples de data binding é estabelecida usando um ‘bigode’ `{{ }}`
 - Também chamada de interpolação
 - Vincula o dado com o texto do elemento HTML
 - Não é aplicável em atributos HTML, para isso diretivas são utilizadas

Criando uma aplicação Vue

Data binding, interpolação e Diretivas

```
const vm = createApp({
  setup() {
    const seen = ref(true)
    return { seen }
  }
}).mount('#app')
vm.seen = false
```

```
<p v-if="seen">Now you see me</p>
```

Now you see me

- **Diretivas** são atributos especiais começados com o prefixo **v-**
- Utilizadas para realizar a vinculação de atributos
- Tem como objetivo aplicar reativamente os efeitos colaterais da mudança dos valores de suas expressões na DOM

Diretivas



Diretivas

Argumentos

- Diretivas são atributos especiais começados com o prefixo **v-**
- Algumas diretivas aceitam um argumento
 - Após o nome da diretiva e um sinal de :
- Por exemplo: v-bind é utilizado para atualizar de forma reativa um atributo HTML

```
<a v-bind:href="url"> ... </a>  
<a v-on:click="doSomething"> ... </a>  
<a v-bind:[attributeName]="url"> ... </a>
```

Diretivas

Renderização condicional

```
<h1 v-if="awesome">Vue is awesome!</h1>
<h1 v-else>Oh no 😞</h1>
```

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

```
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
```

Diretivas

Renderização de listas

```
createApp({
  setup() {
    const items = ref([ { message: 'Foo' }, { message: 'Bar' } ])
    return { items }
  }
}).mount('#app')
```

```
<ul>
  <li v-for="item in items">
    {{ item.message }}
  </li>
</ul>
```

Diretivas

Renderização de listas

```
createApp({
  setup() {
    const items = ref([ { message: 'Foo' }, { message: 'Bar' } ])
    return { items }
  }
}).mount('#app')
```

```
<ul>
  <li v-for="(item, index) in items">
    {{ index }} - {{ item.message }}
  </li>
</ul>
```


Diretivas

Vinculação de classes (CSS)

```
createApp({  
  setup() {  
    const isActive = ref(true)  
    const isError = ref(false)  
    return { isActive, isError }  
  }  
}).mount('#app')
```

```
<div :class="{ active: isActive }"></div>  
  
<div  
  class="static"  
  :class="{  
    active: isActive,  
    'text-danger': hasError  
  }"  
></div>
```

Diretivas

Vinculação de classes (CSS)

```
createApp({
  setup() {
    const activeClass = ref("active")
    const errorClass = ref("danger")
    return { activeClass, errorClass }
  }
}).mount('#app')
```

```
<div :class="[activeClass, errorClass]"></div>
```

Diretivas

Tratando eventos

```
createApp({
  setup() {
    const count = ref(0)
    return { count }
  }
}).mount('#app')
```

```
<div id="basic-event">
  <button v-on:click="counter += 1">Add 1</button>
  <p>The button above has been clicked {{ counter }} times.</p>
</div>
```

Diretivas

Tratando eventos

```
createApp({
  setup() {
    const name = ref('Bruno')
    function greet(event) {
      alert('Hello ' + name.value + '!')
      if (event) { alert(event.target.tagName) }
    }
    return { name, greet }
  }
}).mount('#app')
```

```
<div id="event-with-method">
  <button v-on:click="greet">Greet</button>
</div>
```

Diretivas

Tratando eventos

```
createApp({
  setup() {
    function say(something) {
      alert(something)
    }
    return { say }
  }
}).mount('#app')
```

```
<div id="inline-handler">
  <button v-on:click="say('hi')">Say hi</button>
  <button v-on:click="say('what')">Say what</button>
</div>
```


Diretivas

Vinculação de formulários

- Ao utilizamos formulários, frequentemente precisamos sincronizar o dados de entradas a elementos que JavaScript que representem esse estado
- Usando as diretivas básicas que vimos podemos fazer isso assim:

```
<input  
  :value="text"  
  @input="event => text = event.target.value">
```

- Ou simplesmente assim:

```
<input v-model="text">
```

Diretivas

V-model

- A diretiva *v-model* é usada para criar *two-way binding*
- Pode ser usadas com diferentes tipos de elementos de entrada
- Atualiza corretamente o elemento baseado no tipo de *input*
 - *text* e *textarea* usam a propriedade *value* e o evento *input*
 - *checkbox* e *radio buttons* usam a propriedade *checked* e o evento *change*
 - *select* usam a propriedade *value* e o evento *change*

Diretivas

Vinculação de formulários: Input e Textarea

```
<input v-model="message" placeholder="edit me" />
<p>Message is: {{ message }}</p>
```

```
<span>Multiline message is:</span>
<p style="white-space: pre-line;">{{ message }}</p>
<br>
<textarea v-model="message" placeholder="add multiple lines"></textarea>
```

Diretivas

Vinculação de formulários: checkbox

```
<input type="checkbox" id="checkbox" v-model="checked" />
<label for="checkbox">{{ checked }}</label>
```

```
<div id="v-model-multiple-checkboxes">
  <input type="checkbox" id="jack" value="Jack" v-model="checkedNames" />
  <label for="jack">Jack</label>
  <input type="checkbox" id="john" value="John" v-model="checkedNames" />
  <label for="john">John</label>
  <input type="checkbox" id="mike" value="Mike" v-model="checkedNames" />
  <label for="mike">Mike</label>
  <br>
  <span>Checked names: {{ checkedNames }}</span>
</div>
```

Diretivas

Vinculação de formulários: radiobutton

```
<div id="v-model-radiobutton">
  <input type="radio" id="one" value="One" v-model="picked" />
  <label for="one">One</label>
  <br />
  <input type="radio" id="two" value="Two" v-model="picked" />
  <label for="two">Two</label>
  <br>
  <span>Picked: {{ picked }}</span>
</div>
```

```
createApp({
  setup() {
    const picked = ref('')
    return { picked }
  }
}).mount('#app')
```


Diretivas

Vinculação de formulários: select

```
<div id="v-model=select" class="demo">
  <select v-model="selected">
    <option disabled value="">Please select one</option>
    <option>A</option>
    <option>B</option>
    <option>C</option>
  </select>
  <span>Selected: {{ selected }}</span>
</div>
```

```
createApp({
  setup() {
    const selected = ref('')
    return { selected }
  }
}).mount('#app')
```

Diretivas

Atalhos

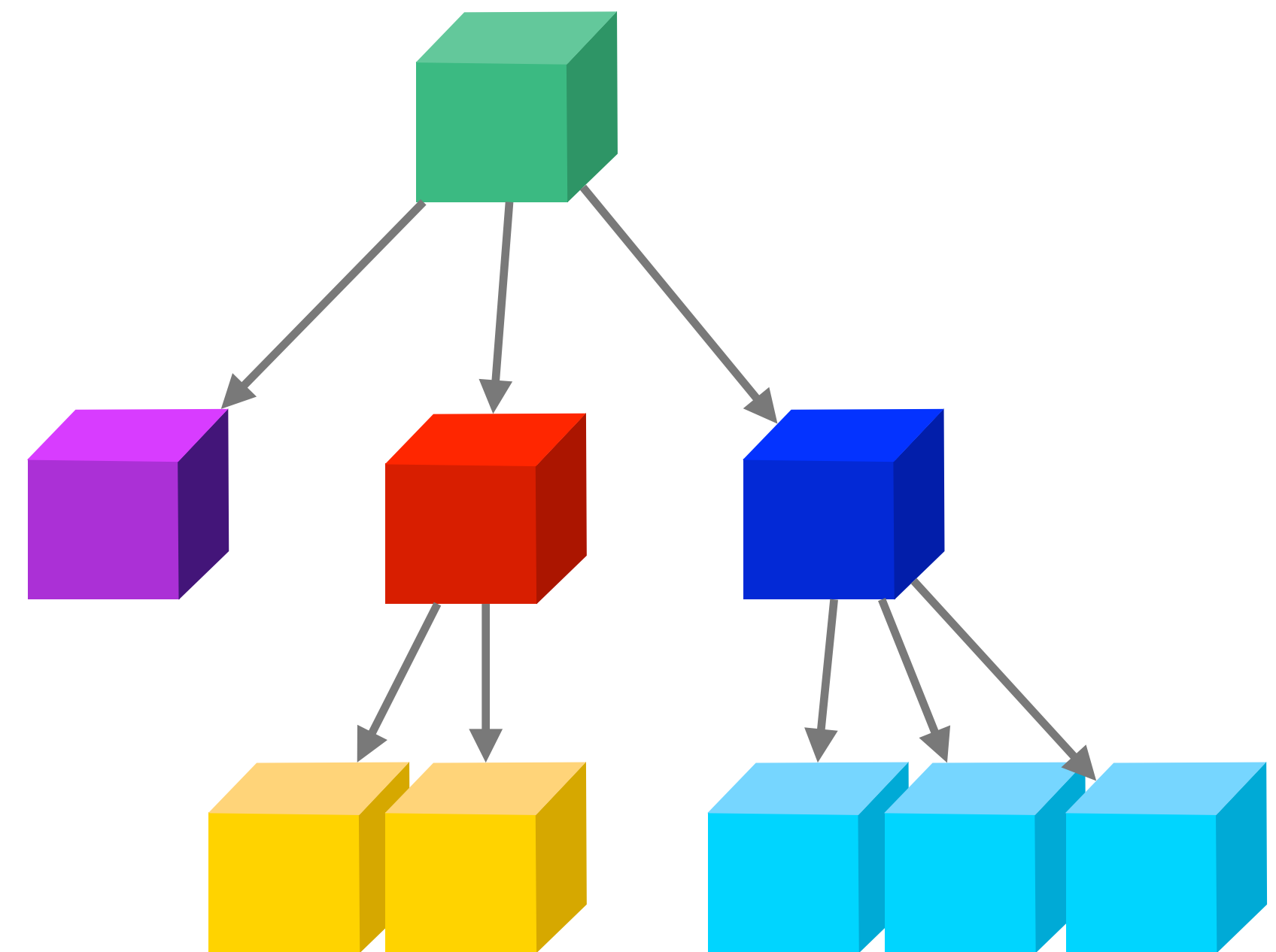
Atalho para v-bind

```
<!-- full syntax -->  
<a v-bind:href="url"> ... </a>  
<!-- shorthand -->  
<a :href="url"> ... </a>  
<!-- shorthand with dynamic argument -->  
<a :[key]="url"> ... </a>
```

Atalho para v-on

```
<!-- full syntax -->  
<a v-on:click="doSomething"> ... </a>  
<!-- shorthand -->  
<a @click="doSomething"> ... </a>  
<!-- shorthand with dynamic argument -->  
<a @[event]="doSomething"> ... </a>
```

Componentes



Componentes

- Assim como outros frameworks modernos, Vue permite que os usuários criem componentes isolados em suas aplicações
- São importantes dentre outros motivos por favorecer a **reusabilidade** e a **manutenabilidade**
- São auto-contidos agrupando HTML, JS e CSS
 - Facilitando a manutenção especialmente quando a aplicação escala

Componentes

Props: Passando dados para um componente filho

- Permite o envio de dados aos components filhos
- São atributos customizáveis registrados por um componente
 - Dever ser explicitamente declarados no componente filho
 - Um valor deve ser dado pelo componente pai/mãe
- São **unidirecionais**, sempre no sentido **pai/mãe -> filho**


Componentes

Passando dados para um componente filho

```
const app = createApp({
  setup() {
    const posts = ref([
      { id: 1, title: 'My journey with Vue' },
      { id: 2, title: 'Bloggng with Vue' },
      { id: 3, title: 'Why Vue is so fun' }
    ])
    return { posts }
  }
})

app.component('blog-post', {
  props: ['title'],
  template: `<h4>{{ title }}</h4>`
})
```

```
<div>
  <blog-post
    v-for="post in posts"
    :key="post.id"
    :title="post.title"
  ></blog-post>
</div>
```



My journey with Vue
Bloggng with Vue
Why Vue is so fun

Componentes

defineModel - Vue 3.4

- Em algumas situações precisamos atualizar o componente pai, a cada atualização de um prop do componente filho
- A partir do Vue 3.4, a maneira recomendada é a seguinte:

```
<script setup>
const model = defineModel()

function update() {
  model.value++
}
</script>

<template>
  <div>Parent bound v-model is: {{ model }}</div>
  <button @click="update">Increment</button>
</template>
```

```
<!-- Parent.vue -->
<Child v-model="countModel" />
```

Componentes

Escutando eventos do componentes filhos

- Em diversas situações é necessário que haja comunicação entre filho e pai
- Essa comunicação é feita por meio de eventos customizados
 - São iniciados quando um componente executa a instrução `$emit('nome do evento')`
 - Um componente que está escutando pelo é evento é notificado na instrução `$on('nome do evento')`
 - Dados podem ser enviados

Componentes

Escutando eventos do componentes filhos

```
const app = createApp({
  setup() {
    const posts = ref([
      { id: 1, title: 'My journey with Vue' },
      { id: 2, title: 'Blogging with Vue' },
      { id: 3, title: 'Why Vue is so fun' }
    ])
    const postFontSize = ref(1)
    return { posts, fontSize }
  }
})
```

```
app.component('blog-post', {
  props: ['title'],
  emits: ['enlargeText'],
  template: `
    <div class="blog-post">
      <h4>{{ title }}</h4>
      <button @click="$emit('enlargeText')">
        Enlarge text
      </button>
    </div>
  `
})
```

```
<div :style="{ fontSize: fontSize + 'em' }">
  <blog-post v-for="post in posts"
    :key="post.id"
    :title="post.title"
    @enlarge-text="fontSize += 0.1"
  ></blog-post>
</div>
```

Componentes

Escutando eventos do componentes filhos

```
const app = createApp({
  setup() {
    const posts = ref([
      { id: 1, title: 'My journey with Vue' },
      { id: 2, title: 'Blogging with Vue' },
      { id: 3, title: 'Why Vue is so fun' }
    ])
    const fontSize = ref(1.0)
    function onEnlargeText(enlargeAmount) {
      fontSize.value += Number(enlargeAmount)
    }
    return { posts, fontSize, onEnlargeText }
  }
})
```

```
app.component('blog-post', {
  props: ['title'],
  emits: ['enlargeText'],
  template: `
    <div class="blog-post">
      <h4>{{ title }}</h4>
      <button @click="$emit('enlargeText', 0.1)">
        Enlarge text
      </button>
    </div>
  `
})
```

```
<div :style="{ fontSize: fontSize + 'em' }">
  <blog-post v-for="post in posts"
    :key="post.id"
    :title="post.title"
    @enlarge-text="onEnlargeText"
  ></blog-post>
</div>
```

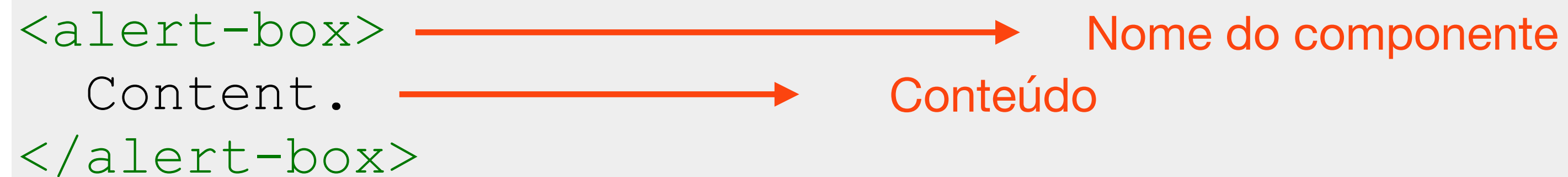
1. Ao clicar no botão o evento **enlargeText** é emitido com o valor **0.1** como argumento
2. O evento é tratado no blog-post
 - O valor **0.1**, é o **valor do parâmetro** recebido
3. o método **onEnlargeText** do **componente pai** é invocado
4. A propriedade do componente pai é atualizada
5. O style do div é atualizado como consequência

Valor enviado

Componentes

Slots

- Assim com elemento HTML, algumas vezes é útil passar o conteúdo para o componente da seguinte forma



The diagram shows the following code snippet with annotations:

```
<alert-box>  
  Content.  
</alert-box>
```

Annotations:

- An orange arrow points from `<alert-box>` to the text "Nome do componente".
- An orange arrow points from `Content.` to the text "Conteúdo".

- Isto pode ser feito ao utilizar o elemento `<slot>`

Componentes

Distribuição de componentes com slots

```
app.component('todo-button', {  
  template: `  
    <button class="btn-primary">  
      <slot></slot>  
    </button>  
  `,  
})
```

```
<todo-button>  
  Add todo  
</todo-button>
```

```
<todo-button>  
  <i class="fas fa-plus"></i>  
  Add todo  
</todo-button>
```

```
<todo-button>  
  <font-awesome-icon name="plus"></font-  
  awesome-icon>  
  Add todo  
</todo-button>
```

Outro componente

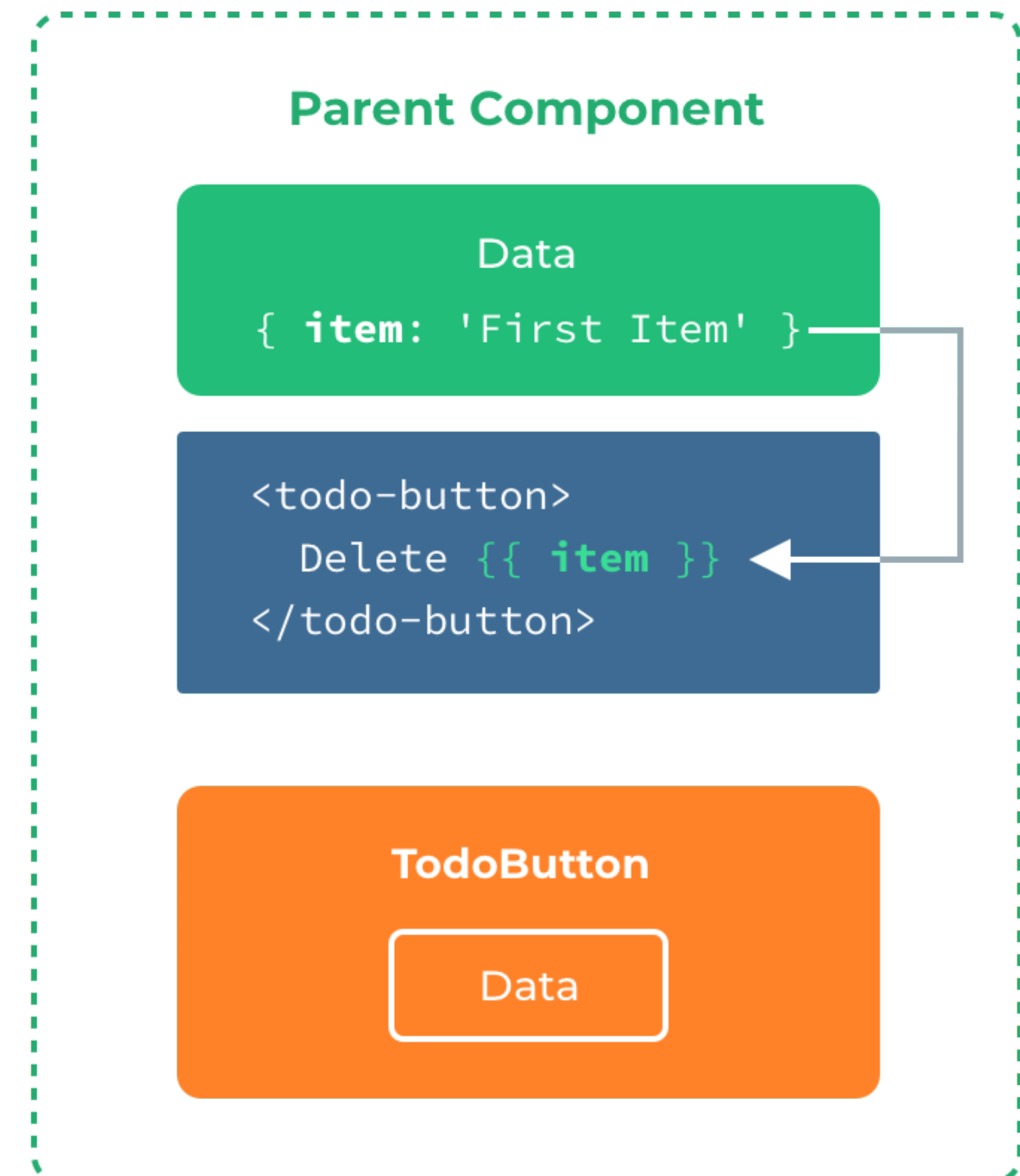
Componentes

Distribuição de componentes com slots

```
app.component('todo-button', {  
  template: `  
    <button class="btn-primary">  
      <slot></slot>  
    </button>`  
})
```

```
<todo-button>  
  Delete a {{ item }}  
</todo-button>
```

Slots não tem acesso a dados do componente filho

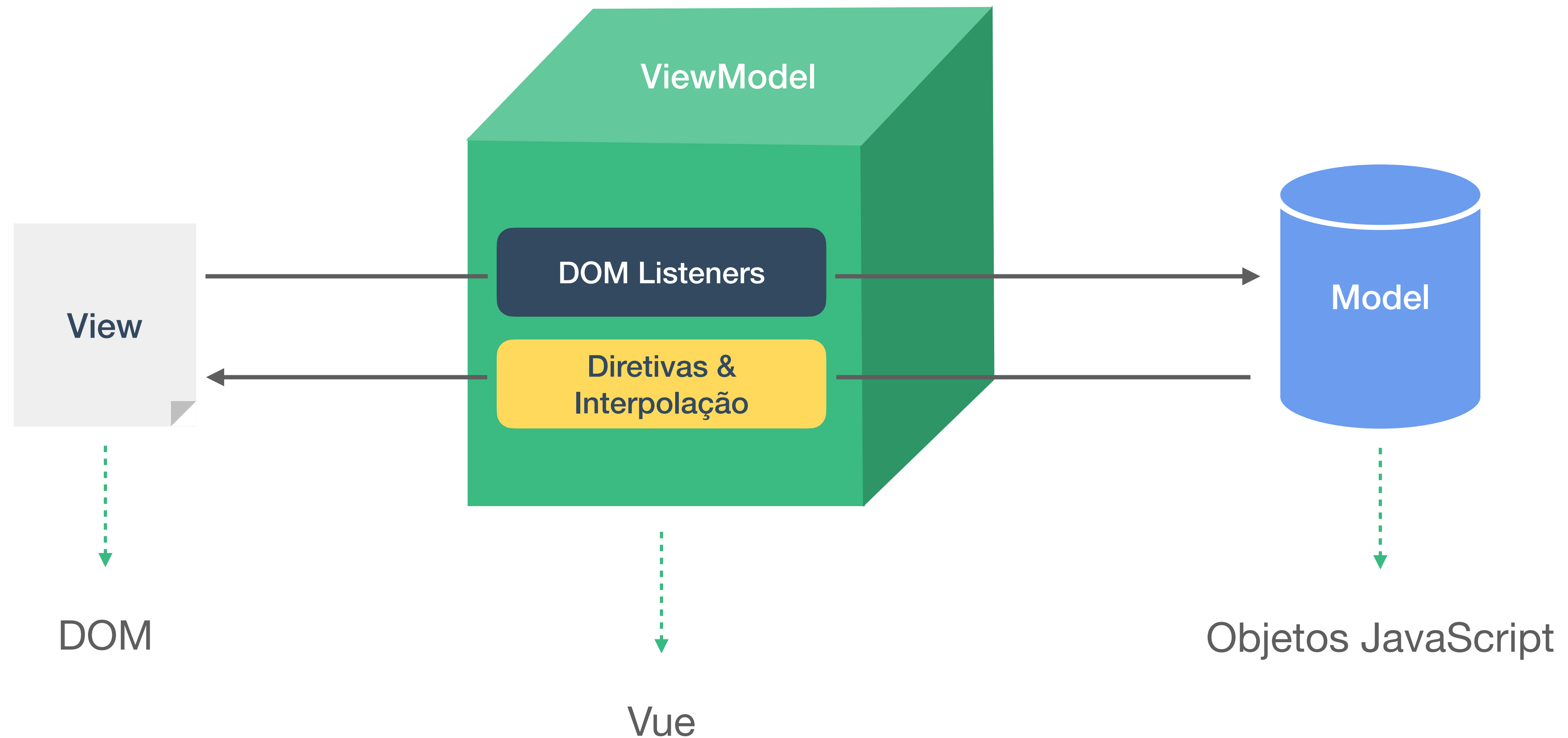


Principais aspectos de uma aplicação em VueJs



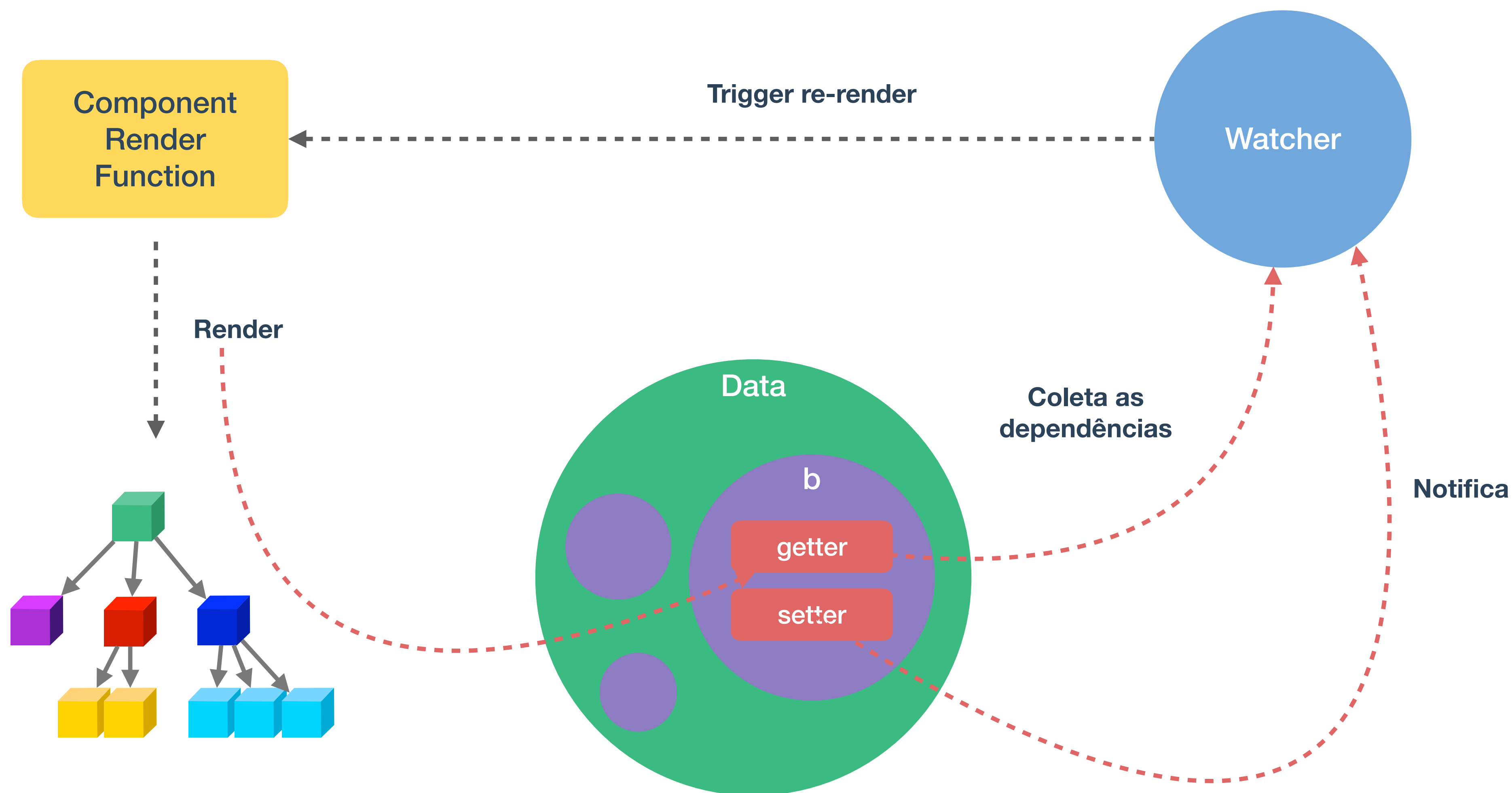
Principais aspectos de uma aplicação em VueJs

Data binding



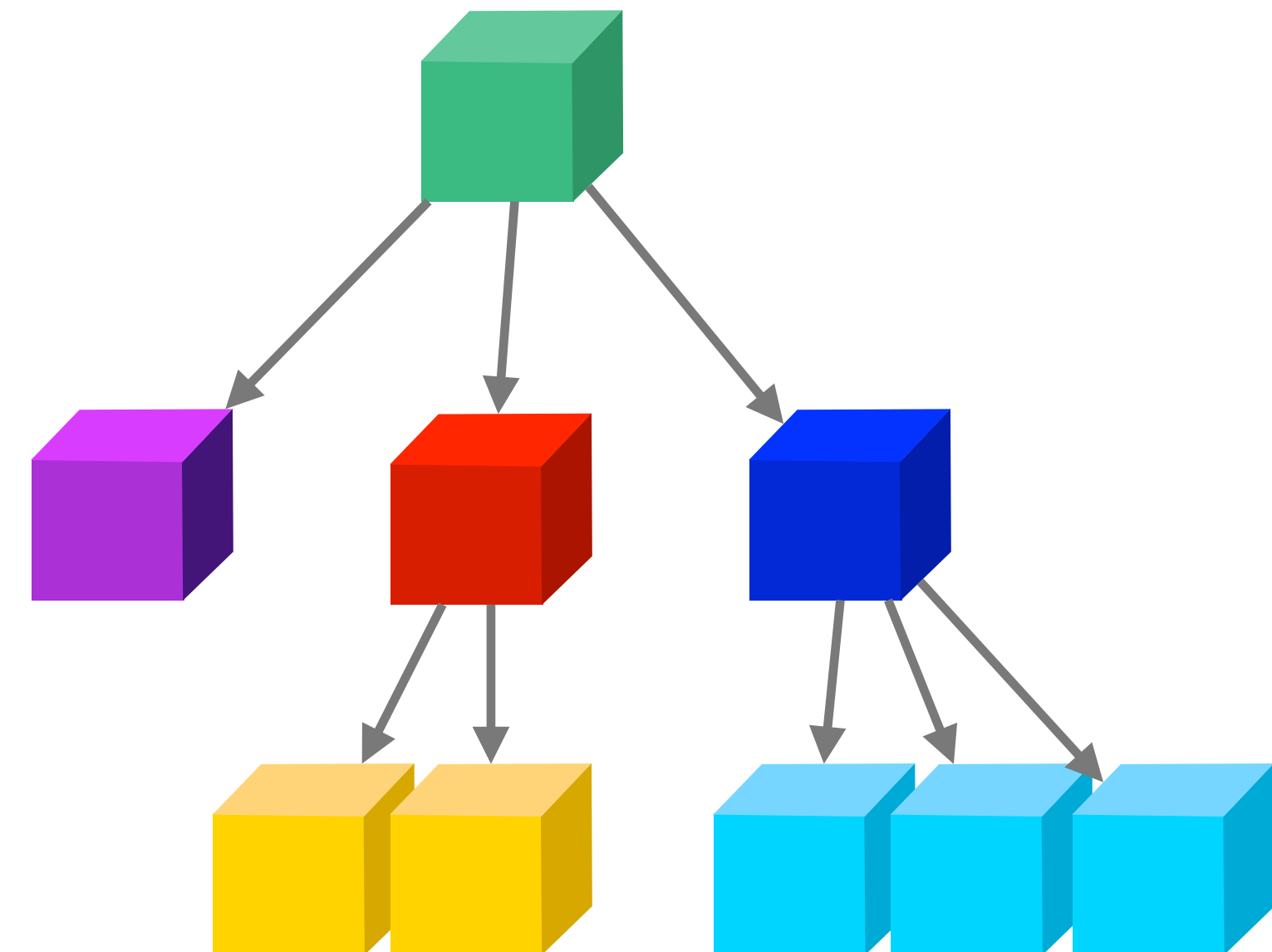
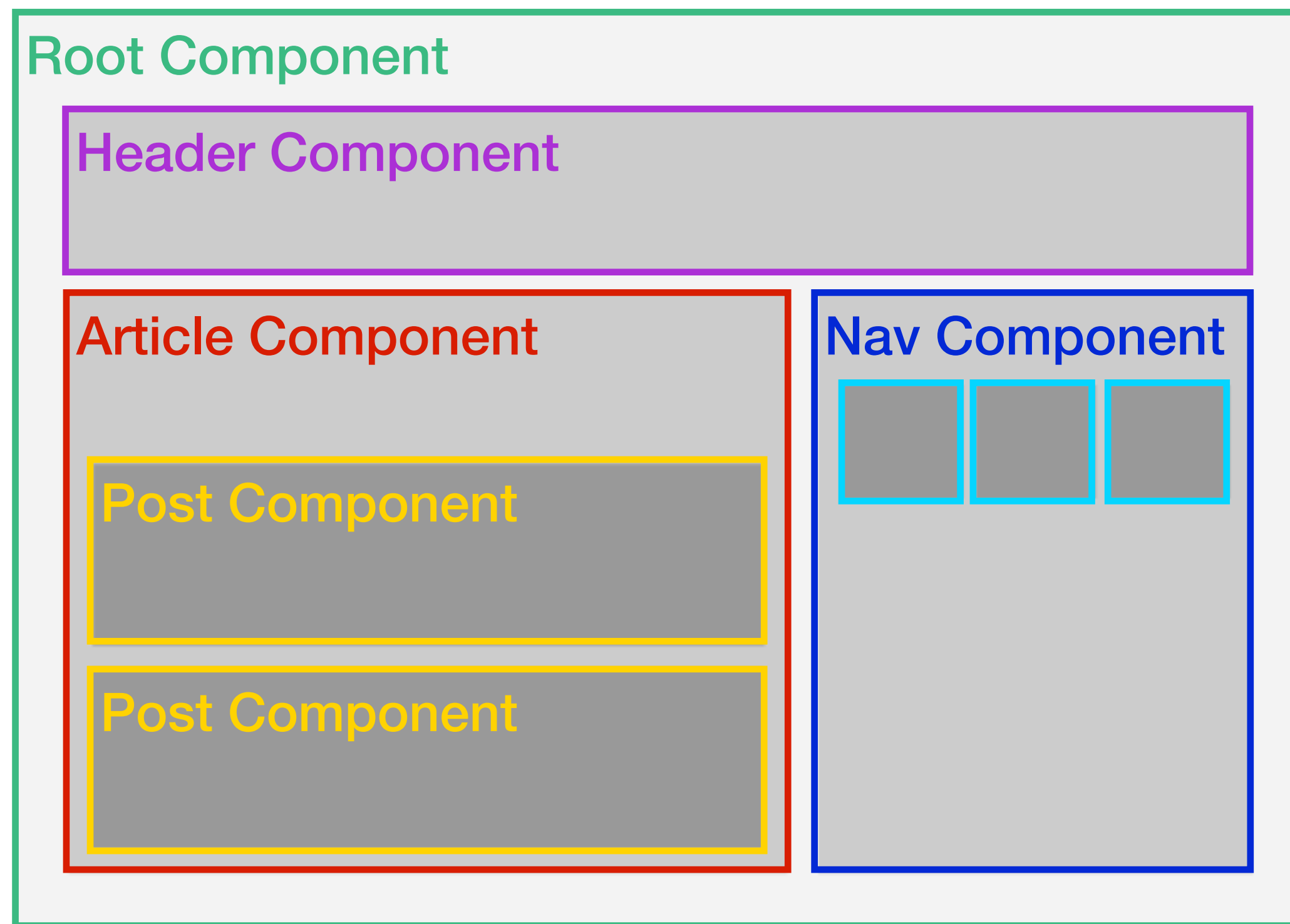
Principais aspectos de uma aplicação em VueJs

Data binding



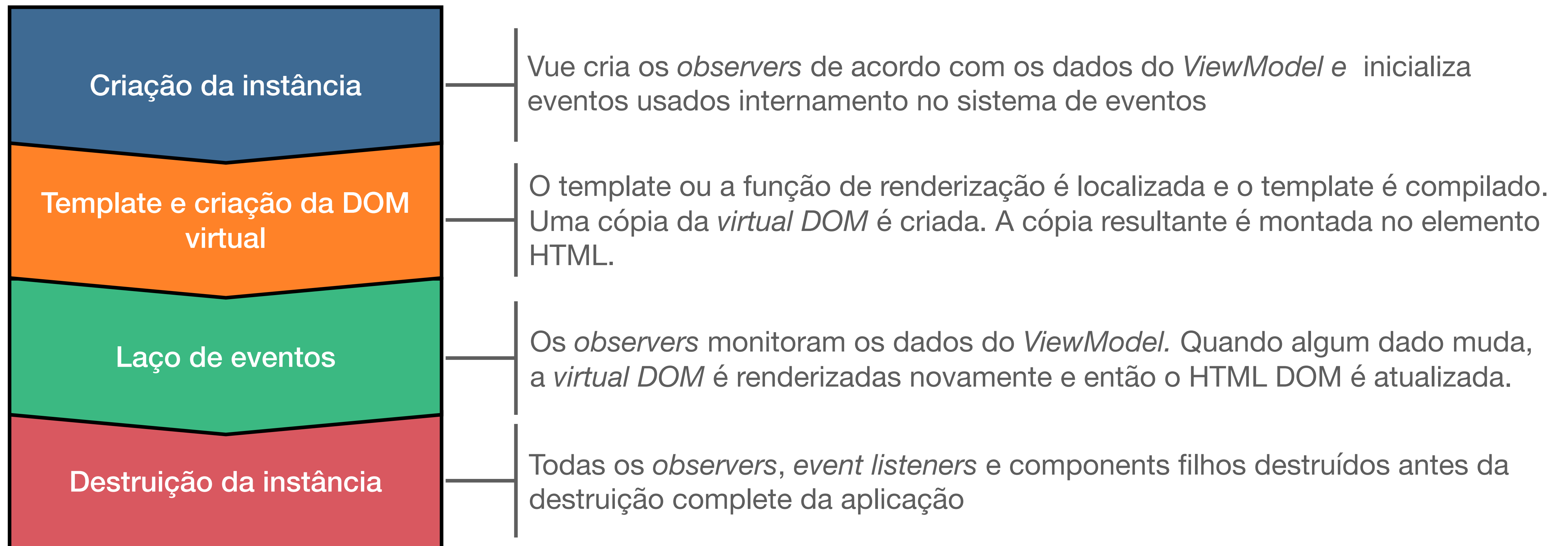
Principais aspectos de uma aplicação em VueJs

Componentes



Principais aspectos de uma aplicação em VueJs

Ciclo de Vida



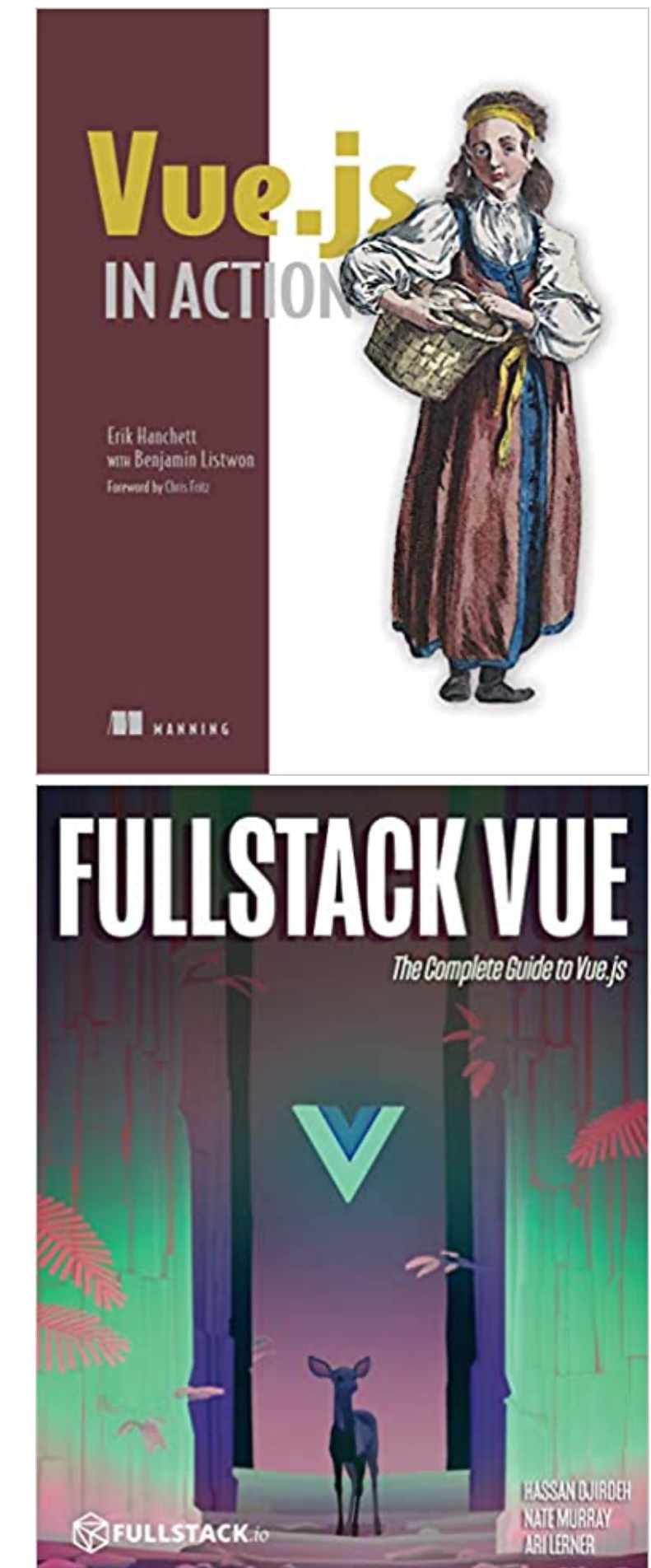
Principais aspectos de uma aplicação em VueJs

Ciclo de Vida

```
app = Vue.createApp(options)  
app.mount(el)
```

Referências

- Vue.js in Action por Eric Hanchett com Benjamin Listwon
- Fullstack Vue: The Complete Guide to Vue.js por Hassan Djirdeh, Nate Murray e Ari Lerner
- A brief history of web development
- FrontEnd Chronology
- Vue Presentation
- Understanding MVVM - A guide for JavaScript developers



Referências

- MVVM - Learning JavaScript Design Patterns [Book]
- Vue.js : Documentação oficial
- The ultimate guid to Javascript frameworks
- JavaScript Technical Interview Question: is React a MVC or MVVM?
- VueJs OverView

Por hoje é só