[Roteiro] Projeto PB4 - Vue.JS

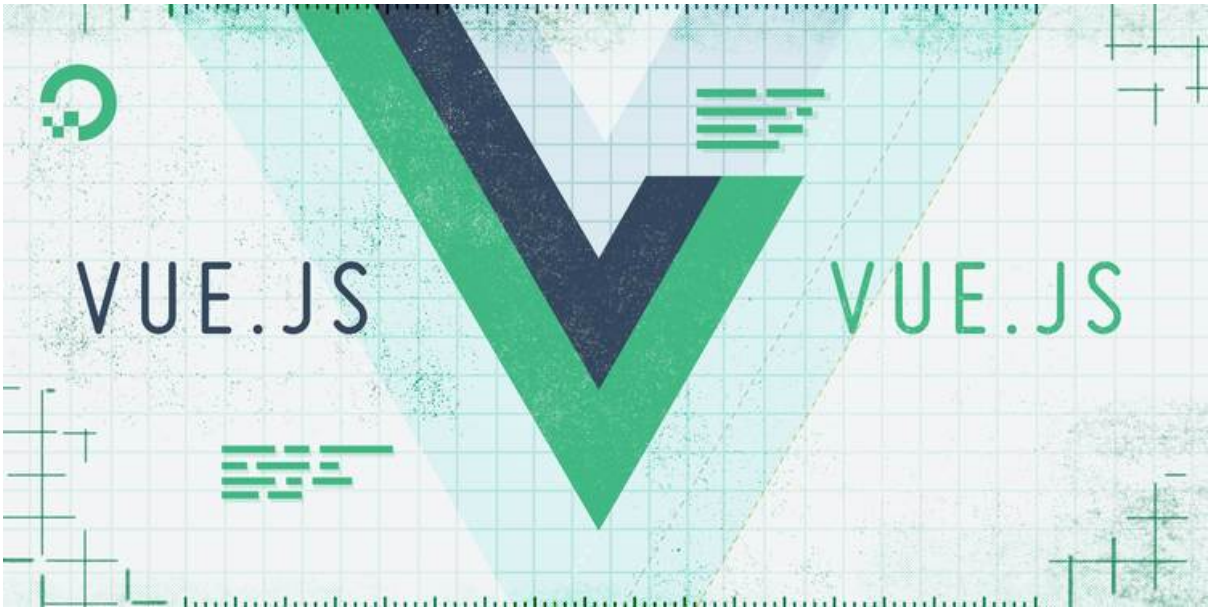
Professor: Johnata Souza Santicioli

INTEGRANTES:

Erik Jonatan Martins Viana - SP3027015

Katharine Fernandes Viana Rodrigues - SP302718X

Larissa Alves de Souza - SP302394X



- Introdução

Nos últimos 10 anos, as páginas da web tornaram-se muito mais dinâmicas e poderosas graças ao uso do JavaScript, de modo que movemos bastante código que normalmente ficaria no servidor para os navegadores, nos deixando com milhares de linhas de código JavaScript conectando vários arquivos HTML e CSS sem nenhuma organização formal.

Por isso cada vez mais e mais desenvolvedores estão usando frameworks JavaScript, como o Angular, React e o Vue, contudo um diferencial do Vue é a maneira com que ele permite, de forma melhor, a organização dos componentes, como eles são estruturados, estilizados e reativos.

Vue é um framework acessível, versátil e performático que ajuda a criar um código mais sustentável e simples de testar

Vue é um framework progressivo, o que significa que ele permite que ainda que você já tenha uma aplicação server-side seja possível incorporar Vue em apenas determinada parte da aplicação que precisar de uma experiência mais rica e interativa.

Por outro lado, se desejamos construir mais lógica de negócios no front-end desde o início, Vue tem as bibliotecas oficiais e o ecossistema necessário para escalar a aplicação com o uso de coisas como Vuex.

Assim como React e Angular, Vue é um framework que trabalha com o conceito de componentes — pequenos elementos de uma página contendo HTML, CSS e JavaScript — que possuem inúmeras vantagens,

desde a capacidade de reutilizar componentes em toda a página até a manutenção do código, que por estar modularizado, permite alterações dinâmicas (alterar o conteúdo do componente muda todas as instâncias daquele componente na página). Esses componentes, por serem feitos unindo as três tecnologias, trivializam tarefas que seriam complexas sem o uso de Vue, como por exemplo, decidir se uma tag HTML deve ser renderizada na página ou não em tempo real.

Por fim, cada componente constitui um escopo isolado dos demais, tanto em lógica quanto nos estilos. A renderização dos dados é feita baseada em uma DOM virtual que é atualizada apenas quando os dados de um componentes são alterados, garantindo eficiência e velocidade.

- História do Vue.Js

O Vue foi criado por Evan You, depois de trabalhar para o Google no AngularJS. Mais tarde, resumiu seu processo de pensamento: "Imaginei — e se eu pudesse extrair a parte que realmente gostava de Angular e construir algo realmente leve sem todos os conceitos extras envolvidos?"

A história começa quando ele estava trabalhando em um dos projetos do Google Creative Labs (iniciativa de experimentação tecnologia da Google). Ele precisava de uma interface de protótipo para UIs bem grandes. Claramente, escrever HTML puro em um projeto de grande porte era consumir tempo e recursos. Pensando nisso, Evan começou a procurar alguma ferramenta já existente para essa finalidade.

Para sua surpresa, descobriu que não havia nenhuma ferramenta, biblioteca ou framework que se adequasse exatamente ao seu propósito

de prototipagem rápida. Naquela época, o ecossistema JS tinha poucos frameworks front-end. O framework que os popularizou, React.js, ainda estava começando, AngularJS era amplamente utilizado, mas tinha uma série de desvantagens para os propósitos de Evan You, e frameworks como Backbone.js eram usados para aplicações em larga escala com arquitetura MVC. Para o tipo de projeto que precisava de algo realmente flexível e leve para uma rápida prototipagem UI, nenhum desses frameworks ajudavam. Foi assim que surgiu a ideia de criar um framework que ajudaria na prototipagem rápida, oferecendo uma maneira fácil e flexível de ligação de dados reativos e componentes reutilizáveis. Agora ele pode ser usado para construir complexos aplicativos escaláveis e reativos na web.

O Vue foi originalmente lançado em fevereiro de 2014. O projeto foi postado no Hacker News, Echo JS e no subreddit r/javascript no dia da sua versão inicial. Dentro de um dia, o projeto alcançou a primeira página dos três sites.

Em 2016, o Vue foi apresentado como uma grande promessa no GitHub, tendo obtido a maioria das estrelas de qualquer projeto de código aberto no popular site, registrando 57,908 estrelas, e figurando entre os projetos open source mais populares no GitHub e a segunda estrutura/biblioteca de JavaScript mais popular (após React).

- Vantagens e desvantagens

Vantagens

- **Acessível:** com um conhecimento básico em HTML, CSS e JavaScript, você já consegue construir uma aplicação básica, já que os componentes são puramente baseados nessas tecnologias;
- **Versátil:** possui um ecossistema incremental e progressivo, oferecendo soluções prontas e padronizadas para a grande maioria das situações;
- **Excelente desempenho;**
- **Progressivo:** se você já tiver uma aplicação server-side por exemplo, você pode utilizar o Vue em apenas uma parte da aplicação, utilizando inclusive somente os módulos que sejam de fato necessários. Ele também se integra muito bem a outros frameworks, não limitando a aplicação a usar apenas uma solução.

Desvantagens

- **Complexidade do paradigma reativo:** este framework "leva a sério" os princípios da programação reativa, o que pode trazer um pouco de complexidade no começo;

  - O que é programação reativa?

  É um modelo ou um paradigma de programação criado inicialmente pela Microsoft que é orientado a fluxo de dados e propagações de estados.

  Estes fluxos de dados (que também são chamados de streams) são em grande parte assíncronos, ou seja, as operações

são independentes umas das outras e não precisam ser executadas em uma sequência específica. Todas as ações quando falamos sobre programação reativa são transmitidas e detectadas por um fluxo de dados, como eventos, mensagens, chamadas e até mesmo as falhas.

Aplicações reativas, então, são constituídas por reações a alterações nestes fluxos de dados.

- **Excesso de flexibilidade:** o Vue.js é um framework extremamente flexível e não muito opinativo, o que pode trazer problemas de falta de padronização de código em algumas equipes e em projetos de maior escala;
- **Recursos um pouco mais limitados em relação a outros frameworks SPA:** embora o Vue.js seja querido e conte com uma comunidade bem ativa, oferecendo uma série de soluções para os problemas mais comuns; o Vue.js ainda não conta com a mesma quantidade de recursos (como plugins) que seus concorrentes diretos possuem – o Angular e o React.

Uma comparação mais detalhada de cada uma das vantagens e desvantagens do framework pode ser encontrado em seu próprio site, em um artigo escrito pela equipe que criou o framework: https://vuejs.org/v2/guide/comparison.html.

- Primeiros passos com a ferramenta

A forma mais simples de integrar o Vue.js em seu projeto é criando um arquivo .html e incluindo Vue com alguma dessas duas opções abaixo:

```
<!-- versão de desenvolvimento, inclui avisos úteis no console  -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
```

ou:

```
<!-- versão de produção, otimizada para tamanho e velocidade -->
<script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
```

Há também a possibilidade de se instalar o Vue para começar o uso de `vue-cli`, usando Node.js, NPM e/ou Yarn. Contudo para iniciantes com o framework não é muito indicado, uma vez que geralmente não se está familiarizado com ferramentas de build baseadas em Node.js.

- Na Prática

Renderização Declarativa

No núcleo do Vue.js está um sistema que nos permite declarativamente renderizar dados no DOM (Document Object Model) usando uma sintaxe de template simples:

```
<div id="app">
  {{ message }}
</div>
var app = new Vue({
  el: '#app',
  data: {
    message: 'Olá Vue!'
  }
})
```

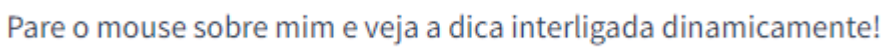
Olá Vue!

Acabamos de criar nosso primeiro aplicativo Vue! Isso parece muito similar a renderizar uma template string, mas Vue fez bastante trabalho interno. Os dados e o DOM estão agora interligados e tudo se tornou reativo. Como podemos ter certeza? Apenas abra o console JavaScript de seu navegador (agora mesmo, nesta página) e atribua um valor diferente em `app.message`. Você verá o exemplo renderizado acima se atualizando de acordo.

Perceba que não temos mais que interagir diretamente com o HTML. Um app Vue acopla-se a um único elemento da DOM (`#app` no nosso caso) e então o controla completamente. O HTML é o nosso ponto de entrada, mas todo o resto acontece dentro da recém criada instância do Vue.

Além de simples interpolação de texto, podemos interligar atributos de elementos:

```
<div id="app-2">
  <span v-bind:title="message">
    Pare o mouse sobre mim e veja a dica interligada dinamicamente!
  </span>
</div>
var app2 = new Vue({
  el: '#app-2',
  data: {
    message: 'Você carregou esta página em ' + new Date().toLocaleString()
  }
})
```



Aqui nos deparamos com algo novo. O atributo `v-bind` que você está vendo é chamado de diretiva. Diretivas são prefixadas com `v-` para indicar que são atributos especiais providos pelo Vue, e como você deve ter percebido, aplicam comportamento especial de reatividade ao DOM renderizado. Neste caso, basicamente está sendo dito: "mantenha o atributo `title` do elemento sempre atualizado em relação à propriedade `message` da instância Vue".

Se você abrir seu console JavaScript novamente e informar `app2.message = 'alguma nova mensagem'`, novamente poderá ver que o HTML vinculado - neste caso, o atributo `title` - foi atualizado imediatamente.

Condicionais e Laços

Também é fácil alternar a presença de um elemento:

```
<div id="app-3">
  <p v-if="seen">Agora você me viu</p>
</div>

var app3 = new Vue({
  el: '#app-3',
  data: {
    seen: true
  }
})
```

Agora você me viu

Vá em frente e informe `app3.seen = false` no console. Você verá a mensagem desaparecer.

Este exemplo demonstra que nós podemos interligar dados não apenas ao texto e aos atributos, mas também à estrutura do DOM. Mais do que isso, Vue também provê um poderoso sistema de transições que pode automaticamente aplicar efeitos de transição quando elementos são inseridos/atualizados/removidos pelo Vue.

Existem mais algumas diretivas, cada uma com sua própria funcionalidade. Por exemplo, a diretiva `v-for` pode ser usada para exibir uma lista de itens usando dados de um Array:

```
<div id="app-4">
```

```
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>

var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
      { text: 'Aprender JavaScript' },
      { text: 'Aprender Vue' },
      { text: 'Criar algo incrível' }
    ]
  }
})
```

1. Aprender JavaScript
2. Aprender Vue
3. Criar algo incrível

No console, informe `app4.todos.push({ text: 'Novo item' })`. Você verá um novo item ser acrescentado dinamicamente à lista.

Tratando Interação do Usuário

Para permitir aos usuários interagir com o aplicativo, podemos usar a diretiva v-on para anexar event listeners que invocam métodos em nossas instâncias Vue:

```
<div id="app-5">
 <p>{{ message }}</p>
 <button v-on:click="reverseMessage">Inverter Mensagem</button>
</div>

var app5 = new Vue({
 el: '#app-5',
 data: {
  message: 'Olá Vue!'
 },
 methods: {
  reverseMessage: function () {
   this.message = this.message.split('').reverse().join('')
  }
 }
})
```

Olá Vue!

Inverter Mensagem

!euV álO

Inverter Mensagem

Observe que neste método atualizamos o estado da aplicação sem tocar no DOM - todas as manipulações são tratadas pelo Vue, o código que você escreve é focado na lógica de manipulação de dados.

Vue também provê a diretiva `v-model`, que torna a interligação de mão dupla (two-way binding) entre a caixa de texto e o estado da aplicação uma moleza:

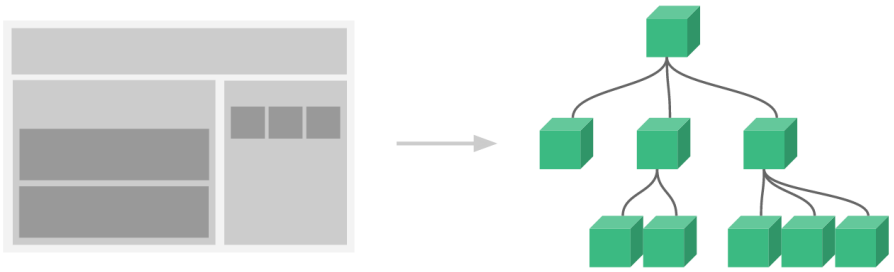
```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>

var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Olá Vue!'
  }
})
```

Olá Vue!

Olá Vue!

Composição com Componentes

O sistema de componentes também é outro importante conceito no Vue, por ser uma abstração que proporciona a construção de aplicações de larga escala compostas por pequenos componentes, auto-contidos e frequentemente reutilizáveis. Se nós pensarmos sobre isso, quase qualquer tipo de interface de uma aplicação pode ser abstraída em uma árvore de componentes:



No Vue, um componente é essencialmente uma instância Vue com opções predefinidas. Registrar um componente no Vue é simples:

```
// Define um novo componente chamado todo-item
Vue.component('todo-item', {
  template: '<li>Isso é um item</li>'
})

var app = new Vue(...)
```

Agora você pode compor com isto no template de outro componente:

```
<ol>
```

```
<!-- Cria uma instância do componente todo-item -->
<todo-item></todo-item>
</ol>
```

Mas isto renderizaria o mesmo texto toda vez que um item fosse utilizado, o que não é lá muito interessante. Devemos poder passar os dados do escopo superior (parent) para os componentes filhos. Vamos modificar o componente para fazê-lo aceitar uma prop:

```
Vue.component('todo-item', {
  // O componente todo-item agora aceita uma
  // "prop", que é como um atributo personalizado.
  // Esta propriedade foi chamada de "todo".
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})
```

Agora podemos passar o dado `todo` em cada repetição de componente usando `v-bind`:

```
<div id="app-7">
  <ol>
    <!--
      Agora provemos cada todo-item com o objeto todo que ele
      representa, de forma que seu conteúdo possa ser dinâmico.
      Também precisamos prover cada componente com uma "chave",
      a qual será explicada posteriormente.
    -->
    <todo-item
```

```
    v-for="item in groceryList"
    v-bind:todo="item"
    v-bind:key="item.id"
  ></todo-item>
 </ol>
</div>

Vue.component('todo-item', {
 props: ['todo'],
 template: '<li>{{ todo.text }}</li>'
})

var app7 = new Vue({
 el: '#app-7',
 data: {
  groceryList: [
   { id: 0, text: 'Vegetais' },
   { id: 1, text: 'Queijo' },
   { id: 2, text: 'Qualquer outra coisa que humanos podem comer' }
  ]
 }
})
```

1. Vegetais
2. Queijo
3. Qualquer outra coisa que humanos podem comer

Este é um exemplo fictício, mas conseguimos separar nossa aplicação em duas pequenas unidades, sendo que o componente filho está razoavelmente bem desacoplado do componente pai graças à

funcionalidade de `props`. Podemos agora melhorar nosso componente `<todo-item>` com template e lógica mais complexos, sem afetar o restante.

Em uma aplicação grande, é essencial dividir todo o aplicativo em componentes para tornar o desenvolvimento gerenciável. Falaremos mais sobre componentes futuramente neste guia, mas aqui está um exemplo (imaginário) da aparência que o template de um aplicativo poderia ter com o uso de componentes:

```
<div id="app">
  <app-nav></app-nav>
  <app-view>
    <app-sidebar></app-sidebar>
    <app-content></app-content>
  </app-view>
</div>
```