**Frameworks front end**

**Temos varios Frameworkse Bibliotecas dicsponiveis para adoção em projetos de aplicações web. Considerandno os que se baseiam em JS, podemos citar:**

* Vue.js
* Angular
* React
* **Vue.js:** 
  + É um framework progressivo para a construção de interfaces de usuário. Ao contrário 7 de outros frameworks monolíticos, ou seja, frameworks que não podem ser integrados com outros, foi projetado desde sua concepção para ser utilizado de modo incremental, isto é, apenas parte do framework pode ser adotada.
  + Basicamente, o Vue mantém uma função que “observa” um objeto JavaScript e reflete qualquer mudança do seu estado no modelo de documento de objeto (Document Object Model – DOM) da página em linguagem de marcação de hipermídia (HTML).
  + [**https://br.vuejs.org/v2/guide/index.html**](https://br.vuejs.org/v2/guide/index.html)
* Componente de arquivo único contém três trechos de código:
  + Template (HTML); folha de estilo em cascata (CSS); JavaScript.

Primeiros passos:

Você pode [**criar um arquivo index.html**](https://github.com/vuejs/vuejs.org/blob/master/src/v2/examples/vue-20-hello-world/index.html) e incluir Vue com:

<!-- 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>

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:

Ex: FrameworksFrontEnd/MeuExercicioVue/index.html

HTML:

<div class="container" id="app">

    <header>

       <h1>{{name}}</h1>

       <div class="avatar"><img src="imgs/avatar.jpg" alt=""></div>

   </header>

</div>

JS:

*var* app = new Vue({

        el: '#app', //elemento (id)

        data:  { //para substituir os dados

            name: 'LaryssaTGS',

        }

});

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:

HTML:

<div id="app-2">

  <span v-bind:title="message">

    Pare o mouse sobre mim e veja a dica interligada dinamicamente!

  </span>

</div>

JS:

*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](https://br.vuejs.org/v2/guide/index.html#Condicionais-e-Lacos)

HTML:

<div id="app-3">

  <p v-if="seen">Agora você me viu</p>

</div>

JS:

*var* app3 = new Vue({

  el: '#app-3',

  data: {

    seen: true

  }

})

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**](https://br.vuejs.org/v2/guide/transitions) 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:

HTML:

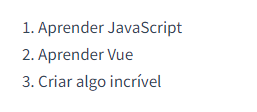
<div id="app-4">

  <ol>

    <li v-for="todo in todos">

      {{ todo.text }}

    </li>

  </ol>

</div>

JS:

*var* app4 = new Vue({

  el: '#app-4',

  data: {

    todos: [

      { text: 'Aprender JavaScript' },

      { text: 'Aprender Vue' },

      { text: '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](https://br.vuejs.org/v2/guide/index.html#Tratando-Interacao-do-Usuario)

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

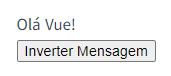
HTML:

<div id="app-5">

  <p>{{ message }}</p>

  <button v-on:click="reverseMessage">Inverter Mensagem</button>

</div>

JS:

*var* app5 = new Vue({

  el: '#app-5',

  data: {

    message: 'Olá Vue!'

  },

  methods: {

    reverseMessage: *function* () {

      this.message = this.message.split('').reverse().join('')

    }

  }

})

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:

HTML:

<div id="app-6">

  <p>{{ message }}</p>

  <input v-model="message">

</div>

JS:

*var* app6 = new Vue({

  el: '#app-6',

  data: {

    message: 'Olá Vue!'

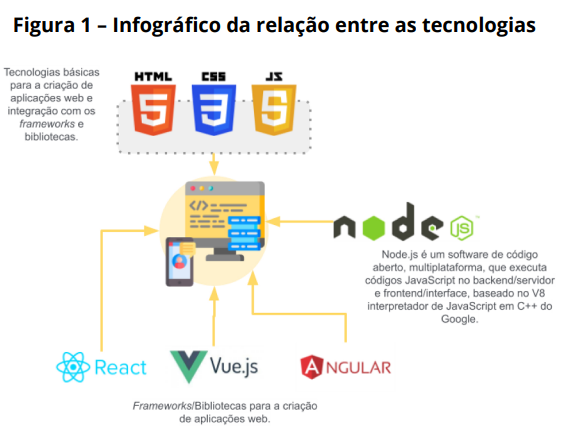
  }

})

* **Angular:**
  + É um framework de código aberto usado para construir aplicativos para web baseados em uma única página dinâmica. Os desenvolvedores também usam essa ferramenta para criar menus animados para páginas de internet baseadas em [**HTML**](https://www.hostinger.com.br/tutoriais/o-que-e-html-conceitos-basicos).
  + O AngularJS usa a arquitetura [**Model-View-Controller**](https://pt.wikipedia.org/wiki/MVC) (MCV), que é usado no desenvolvimento de apps para a internet. Esse tipo de arquitetura consiste em:
    - **Model**(Modelo) – a estrutura de dados que gerencia a informação e recebe comandos do controle.dir
    - **View**(Visão) – a representação da informação.
    - **Controller**(Controle) – responde aos comandos e interage com o modelo.

No contexto do AngularJS, o modelo é o framework, enquanto a visão é o HTML e o controle é o JavaScript. Assim, o AngularJS também pode ser chamado de framework Angular ou Angular Java.

* [**https://www.hostinger.com.br/tutoriais/o-que-e-angular**](https://www.hostinger.com.br/tutoriais/o-que-e-angular)
* **React**
  + O ReactJS é uma das bibliotecas [**de JavaScript**](https://www.hostinger.com.br/tutoriais/o-que-e-javascript) mais populares para desenvolvimento de aplicativos web ou para dispositivos móveis. Criado pelo Facebook, o React contém uma coleção de trechos de código de JavaScript reutilizáveis chamados componentes, usados para a construção da interface do usuário (UI).
  + É importante observar que o ReactJS não é um framework JavaScript. Isso porque ela é responsável apenas por renderizar os componentes da camada de visualização de uma aplicação. O React é uma alternativa a frameworks como [**Angular**](https://www.hostinger.com.br/tutoriais/o-que-e-angular) e Vue, que permitem criar funções complexas.
  + [**https://www.hostinger.com.br/tutoriais/o-que-e-react-javascript**](https://www.hostinger.com.br/tutoriais/o-que-e-react-javascript)

****