Plataformas de aplicações Web

Aula 08 - Comportamento reativo com o Vue.js

Material Didático do Instituto Metrópole Digital - IMD

Apresentação

Nessa aula vamos explorar o que é o comportamento reativo, como ele pode ajudar na criação de páginas dinâmicas e como isso pode aumentar sua produtividade e experiência do usuário.

Vamos ver sua definição, além de um exemplo simples utilizando uma biblioteca javascript reativa para implementar o mesmo projeto da aula passada, porém de forma reativa.

Recapitulando

Vimos na aula passada um exemplo de um comportamento dinâmico em Javascript puro em um sistema com um HTML que consultava dados JSON providos pela aplicação de forma assíncrona e alterava o estado de uma lista UL de animais com os novos dados recebidos em JSON.

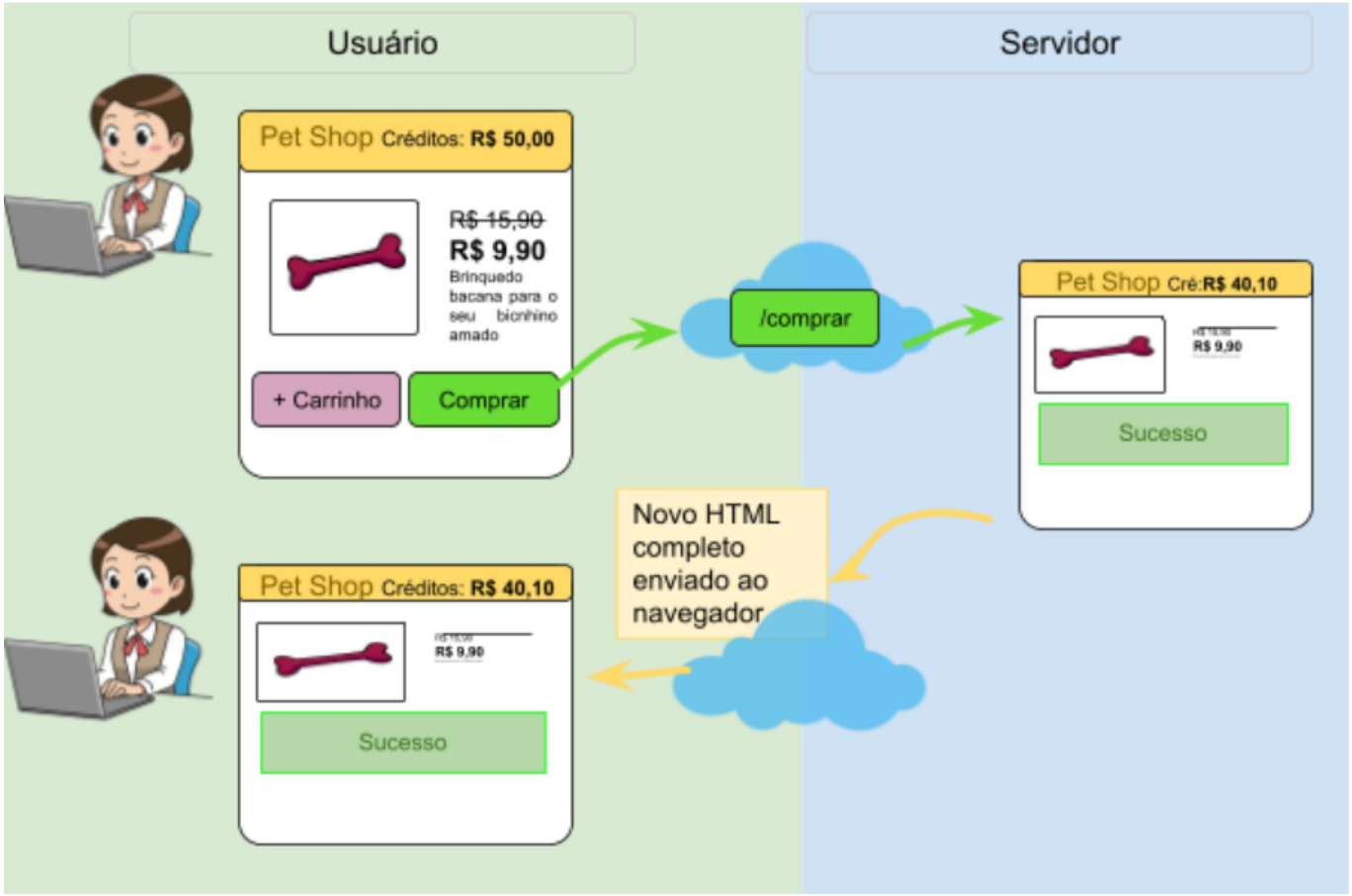
Para obter tal comportamento criamos um sistema Express que servia um documento chamado animais.html na rota "/animais" e esse documento tinha um HTML+Javascript que realizava as operações:

- Permitia ao usuário escolher entre Cães e Gatos
- Rodava um código Javascript que consultava /dados/caes.json ou /dados/gatos.json
- Em Javascript, manualmente, limpava a lista de animais e, um a um, eram criados novos elementos LI com nome - raça de cada animal, os adicionando na lista UL.

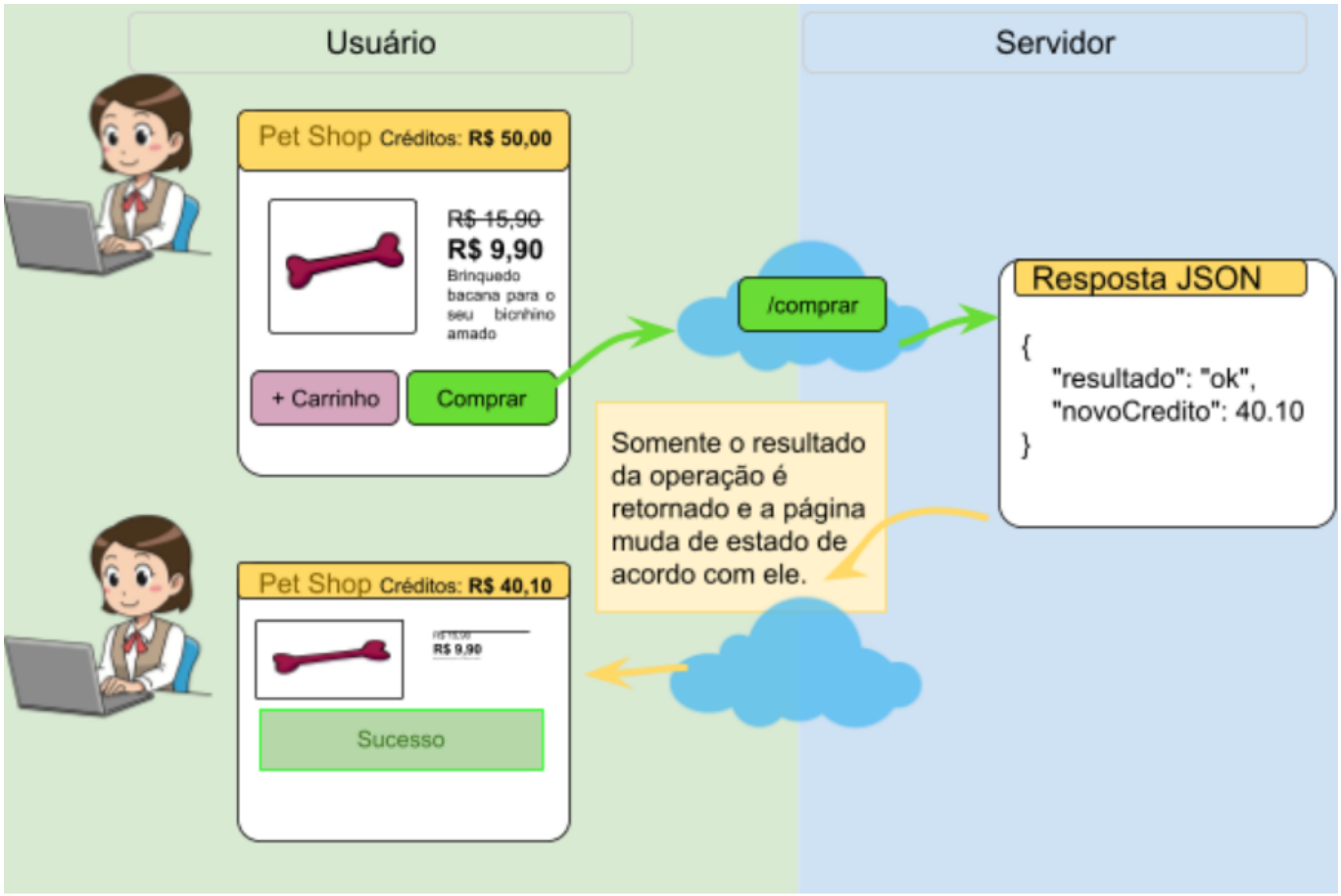
Essa abordagem funciona, mas ela pode ficar mais complicada quando sua página fica maior, com mais elementos e com componentes mais complexos.

Comportamento dinâmico + reativo

Em um sistema web, tradicionalmente o navegador tem um papel mais passivo, de enviar requisições e receber o resultado em um novo documento que substitui o atualmente exibido. Todas as regras de processamento dos dados, modificação das bases de dados, gerenciamento de sessão, geração de documento HTML de resposta, etc, ficam no servidor, como ilustrado abaixo.



Fluxo web tradicional, HTML gerado no servidor



Fluxo dinâmico, HTML inicialmente gerada no servidor, mas alterada no cliente com novos dados.

Nessas imagens acima vemos que no fluxo dinâmico um JSON é retornado e o cliente precisa adaptar o HTML para refletir os novos dados.

Na aula passada essa "adaptação" do HTML foi feita com Javascript puro, alterando elementos no DOM manualmente. Usamos Javascript puro, mas mesmo com o uso do jQuery, ainda estaríamos alterando os elementos manualmente e não reativamente.

Não seria melhor se na nossa página da aula anterior existisse, por exemplo, um array de animais e quando esse array fosse alterado (por exemplo com dados vindo do fetch) a lista de animais no HTML mudasse automaticamente, refletindo os novos dados, sem a necessidade de se limpar ela e a recriar usando Javascript manualmente?

Existem bibliotecas Javascript que podem te ajudar nisso e vamos nessa aula escolher uma para recriar o projeto da aula anterior a utilizando.

Plataformas reativas

Existem bibliotecas que ajudam na criação de páginas dinâmicas com comportamento reativo e com componentes reutilizáveis.

Algumas plataformas front-end de componentes reativas são:

- React (https://reactjs.org/)
- Vue.js (https://vuejs.org/)
- Svelte (https://svelte.dev/)

Todas essas plataformas são baseadas em Javascript e fornecem maneiras diferentes de se ter reatividade nas suas páginas. Iremos entender melhor sobre elas mais a frente na disciplina.

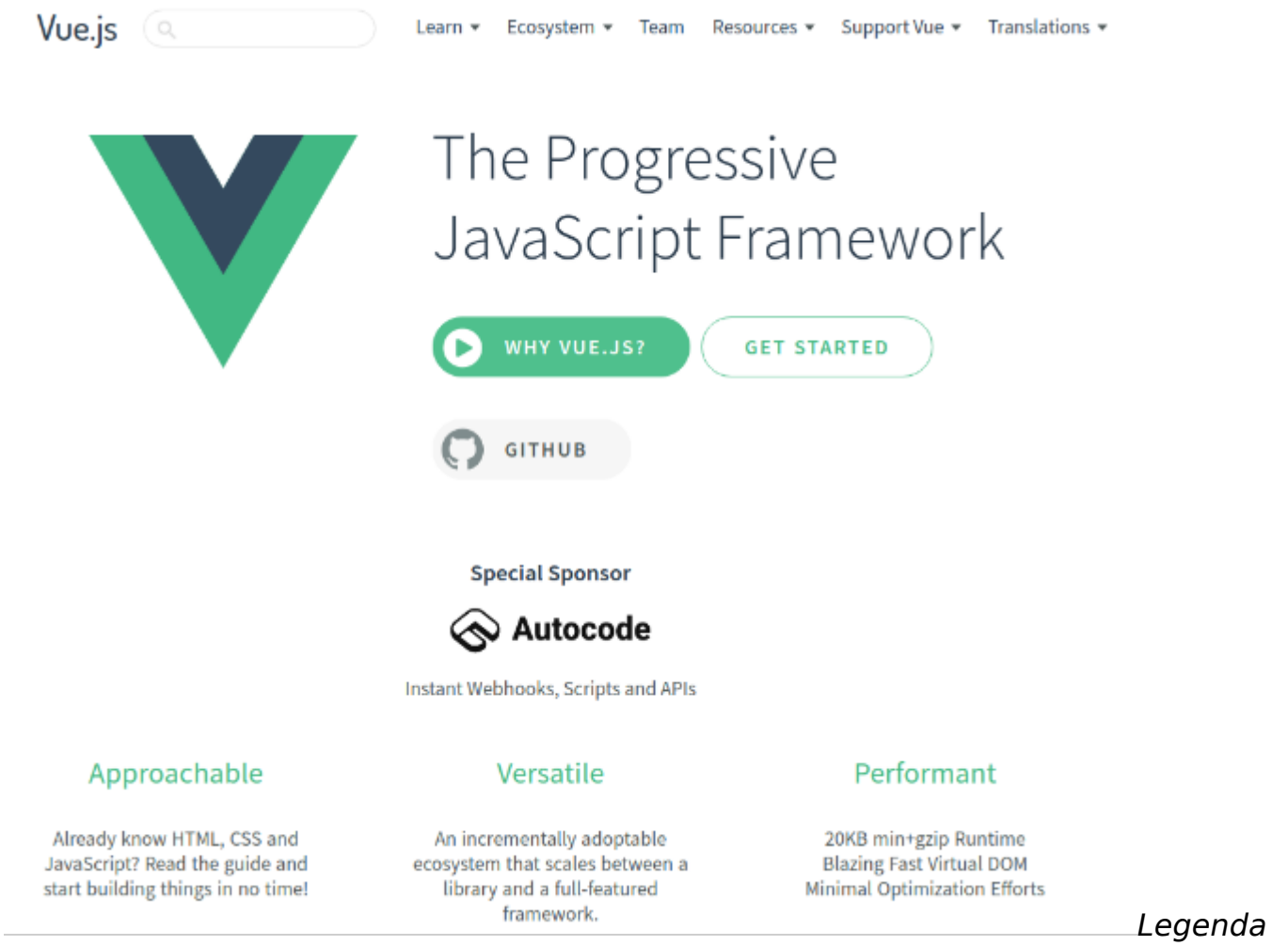
Nessa aula iremos utilizar o Vue.js para recriar o sistema da aula anterior e veremos a diferença de implementação.

Vue.js

Web site: https://vuejs.org/

O Vue.js se define como: "um framework progressivo para a construção de interfaces de usuário. Ao contrário de outros frameworks monolíticos, Vue foi projetado desde sua concepção para ser adotável incrementalmente. A biblioteca principal é focada exclusivamente na camada visual (view layer), sendo fácil adotar e integrar com outras bibliotecas ou projetos existentes. Por outro lado, Vue também é perfeitamente capaz de dar poder a sofisticadas Single-Page Applications

quando usado em conjunto com ferramentas modernas e bibliotecas de apoio.



Legenda da imagem

Página do Vue.js. Fonte: https://vuejs.org/

O Vue (se pronuncia como "view" em inglês), se destaca por ser simples de se integrar em projetos e páginas e por ter reatividade e componentes dinâmicos como pontos forte, fornecendo as mesmas funcionalidades de outros concorrentes como data-bind, two way, eventos, entre outros.

Vamos somente explorar algumas funcionalidades mínimas do Vue nessa aula para que possamos tornar o projeto da aula passada totalmente reativo.

Para mais informações sobre o Vue.js acesse: https://vuejs.org/

Inserindo o Vue.js no projeto

Utilizando o projeto da aula passada, vamos adicionar o Vue.js através de um CDN, para facilitar nossa vida. É possível integrar um Vue em um projeto de diferentes maneiras, como pode ser visto na documentação de instalação em: https://br.vuejs.org/v2/guide/installation.html

Se preferir, antes de alterar o projeto da aula passada, faça uma cópia inteira dele para outra pasta chamada "vueanimais", somente para ter as duas versões na sua máquina.

Na versão Vue da sua aplicação, abra o arquivo em "/pages/animais.html" e adicione a seguinte linha dentro da tag HEAD, logo depois das linhas que adicionam o Bootstrap:

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

Vamos inicialmente "limpar" todo o Javascript e algumas outras coisas na nossa página. Para isso precisamos fazer algumas mudanças.

- Remova todo o javascript atualmente no projeto (inclusive as tags `<script>` e `</script>`
- Remova todos os atributos "id" e "onChange" dos elementos, deixando somente o atributo "class="container" do DIV principal e o "class="form-select" no SELECT.
- No FINAL do HTML, entre as tags `</body>` e `</html>` adicione uma nova tag `<script></script>` que conterá todo o código do Vue. Essa tag ser no final do documento é importante pois o Vue precisa ser executado como o último passo da página, quando o HTML já está pronto. Vamos ainda colocar o código javascript do Vue aí.

Sua página limpa, somente com o Vue adicionado por CDN e sem nenhum Javascript ficará assim:

```
<!DOCTYPE html>

<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstra
p.min.css" rel="stylesheet" integrity="sha384-
KyZXEAg3QhqLMpG8r+8fhAXLRk2vvoC2f3B09zVXn8CA5QIVfZOJ3BCsw2P0p/We"
crossorigin="anonymous">
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/js/bootstrap.
bundle.min.js" integrity="sha384-
```

```
U1DAWAznBHeqEIlVSCgzq+c9gqGAJn5c/t99JyeKa9xxaYpSvHU5awsuZVVFIhvj"
crossorigin="anonymous"></script>
  <script
src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <title>Animais</title>

</head>

<body>
  <div class="container" >
    <h1>Animais</h1>
    <select class="form-select">
      <option value="">Escolha um tipo abaixo</option>
      <option value="caes">Cães</option>
      <option value="gatos">Gatos</option>
    </select>
    <ul>
    </ul>

  </div>

</body>
<script>
<!-- aqui ficará o código do Vue -->
</script>

</html>
```

Sim, perdemos toda a parte dinâmica, mas calma, vamos adicionar novamente usando o Vue.js

Criando o App Vue.js

Sua página está agora com Vue sendo incluído por CDN e removemos todo o Javascript que tinha nela. Agora vamos usar o Vue para buscar os Cães e Gatos reativamente.

O Vue.js precisa criar o que ele chama de "Vue App" para gerenciar os dados de uma aplicação Vue na página.

Primeiramente vamos criar o javascript necessário para Vue gerenciar o estado do HTML da sua página.

Próximo ao fim do documento, após a tag `</body>` e antes da tag `</html>` você

criou uma tag script vazia. Vamos nela colocar o seguinte código, deixando-a assim:

```
<script>
  var app = new Vue({
  el: '#meuAppVue',
  data: {
    tipoAnimal: '',
    animais: []
  },
  methods: {
    async escolherTipoAnimal() {
      if (this.tipoAnimal == '') return;
      const animaisJSONURL = `dados/${this.tipoAnimal}.json`;
      const resultado = await fetch(animaisJSONURL);
      if (resultado.ok) {
        this.animais = await resultado.json();
      }
    }
  }

})

</script>
```

Esse é todo o código Javascript que será necessário no nosso sistema. Vamos analisar.

Primeiro é criado um objeto do tipo Vue chamado "app":

```
var app = new Vue({....resto do do codigo....})
```

Esse app é nosso componente Vue nessa página. Ele recebe algumas configurações. As principais são:

- "el" que informa qual elemento esse componente Vue está gerenciando (no nosso caso é o "#meuAppVue", ou seja, a div com o id="meuAppVue" (que ainda não existe)
- "data" que diz ao Vue qual dados esse componente estará guardando. No nosso caso são dois:
    - A string "tipoAnimal", que pode ser 'caes' ou 'gatos'. Inicia como string vazia.
    - O Array "animais" que guardará a lista de animais que retornará do fetch que busca os documentos json (como anteriormente). Inicia como Array vazio [].

- "methods" tem uma coleção de métodos que iremos executar na nossa aplicação dinamicamente. No nosso caso é só um chamado "escolherTipoAnimal()" que roda quando o select muda de opção. Ele tem que ser declarado como async pois usa usa o await fetch como sintaxe para buscar os elementos. Não se preocupe muito com isso, mas se desejar saber mais basta acessar: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

Basicamente a parte de Javascript é essa. Um app Vue, gerenciando o elemento '#meuAppVue', com dois dados (tipoAnimal e animais) e com um método escolherTipoAnimal.

Repare que o método "escolherTipoAnimal" agora é muito mais simples que anteriormente. Ele simplesmente gera a URL do JSON, mas diferente de antes ele não rpecisa obter o tipo de animal escolhido do elemento usando javascript, e sim somente usa o valor do dado "tipoAnimal" que está declarado no app Vue usando "this.tipoAnimal" para isso (precisa usar o this pois é um dado do app Vue). Depois ele simplesmente usa o fetch para obter o JSON e altera o valor de this.animais que o array do nosso app view. Ele não precisa fazer nenhuma alteração nos elementos HTML, limpar a lista de animais, criar elementos LI, nada disso. Essa parte é tratada de forma diferente com veremos mais a frente.

Essas configurações não estão ainda sendo aplicadas no HTML, elas somente estão prontas para serem utilizadas.

Usando o Vue no HTML

Vamos agora mudar o HTML para usar esses dados configurados no Vue app e chamar a ação correta quando o select mudar de opção.

Para isso vamos fazer poucas mudanças, porém usando as funcionalidades do Vue juntas aos elementos do HTML.

O nosso app Vue foi configurado para gerenciar uma DIV com o ID="meuAppVue". Isso quer dizer que tudo que estiver dentro dessa dig poderá usar os dados e métodos do app Vue criado. Já temos no projeto uma div com a class="container" que é uma excelente escolha para ser também a div que o app Vue vai gerenciar, pois ela engloba todo o HTML do projeto. Vamos então adicionar id="meuAppVue" nessa div, deixando ela assim:

```
<div id="meuAppVue" class="container" >
```

Agora vamos adicionar dois atributos especiais do Vue na tag SELECT dos tipo de animais. Esses atributos são:

- v-model="tipoAnimal"
- v-on:change="escolherTipoAnimal"

O v-model informa para o SELECT que dado ele estará associado. No caso será o dado "tipoAnimal" declarado no nosso app Vue. Isso significa que quando o usuário escolher um novo tipo de animal (caes ou gatos) o valor do dado "tipoAnimal" será automaticamente alterado.

O v-on:change funciona igual ao onChange do HTML comum, porém executará um método que está declarado na configuração "methods" do nosso app Vue. No nosso caso é o "escolherTipoAnimal".

A tag select ficará agora assim:

```
<select v-model="tipoAnimal" v-on:change="escolherTipoAnimal"
class="form-select">
```

Agora vamos fazer com que a lista de animais (UL) seja reativa ao Array "animais" criado dentro do nosso app Vue. Quando a método escolherTipoAnimal alterar esse Array, a tag UL com a lista de animais automaticamente mudará seu conteúdo. Para isso usaremos o atributo especial do Vue chamado "v-for" indicando que essa UL fará um "for" iterando em um array e criando vários elementos dentro dele, porém reativamente, observando as mudanças do array e mudando se precisar, mesmo depois de ter sido gerado pela primeira vez.

O v-for tem a seguinte sintaxe: v-for="item in nomeDoArray". No nsso caso seria v-for="animal in animais". Isso faz com que "dentro" da tag UL possamos usar a variável "animal" representando cada animal listado para criar um LI correspondente. Existe uma linguagem de template para isso no Vue. Ela é bem simples, para mostrar um atributo do animal fazemos algo como: {{ animal.nome }} ou então {{ animal.raca }}.

A tag UL com a lista de animais ficará agora assim:

```
<ul v-for="animal in animais">
     <li>{{ animal.nome }} - {{ animal.raca }}</li>
</ul>
```

Repare que o "UL" funciona agora como um laço que itera em todos os animais criando um LI para cada um. A diferença é que se você alterar o dado "animais" esse código executa novamente, automaticamente mudando a listagem, sem você precisar manipular os seus elementos manualmente, como já dito. Isso é reatividade.

Pronto. Realmente é só isso. Nosso app Vue está criado e o HTML está com os atributos para usar seus dados e executar seus métodos. O HTML com o Vue.js final ficou assim:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstra
p.min.css" rel="stylesheet" integrity="sha384-
KyZXEAg3QhqLMpG8r+8fhAXLRk2vvoC2f3B09zVXn8CA5QIVfZOJ3BCsw2P0p/We"
crossorigin="anonymous">
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/js/bootstrap.
bundle.min.js" integrity="sha384-
U1DAWAznBHeqEIlVSCgzq+c9gqGAJn5c/t99JyeKa9xxaYpSvHU5awsuZVVFIhvj"
crossorigin="anonymous"></script>
  <script
src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <title>Animais</title>

</head>

<body>
  <div id="meuAppVue" class="container" >
    <h1>Animais</h1>
    <select v-model="tipoAnimal" v-on:change="escolherTipoAnimal"
class="form-select">
      <option value="">Escolha um tipo abaixo</option>
      <option value="caes">Cães</option>
      <option value="gatos">Gatos</option>
    </select>

    <ul v-for="animal in animais">
      <li>{{ animal.nome }} - {{ animal.raca }}</li>
    </ul>

  </div>

</body>
```

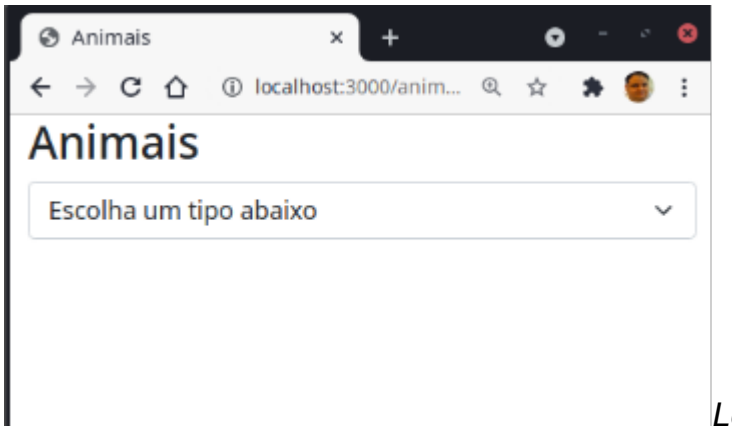
```
<script>
  var app = new Vue({
  el: '#meuAppVue',
  data: {
    tipoAnimal: '',
    animais: []
  },
  methods: {
    async escolherTipoAnimal() {
      if (this.tipoAnimal == '') return;
      const animaisJSONURL = `dados/${this.tipoAnimal}.json`;
      const resultado = await fetch(animaisJSONURL);
      if (resultado.ok) {
        this.animais = await resultado.json();
      }
    }
  }

})

</script>

</html>
```

Bem menor e bem mais simples que a versão com Javascript puro.

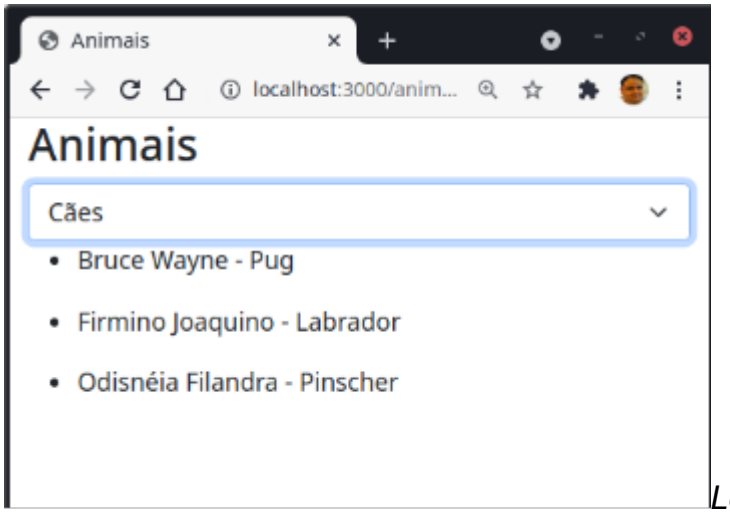
Execute o seu sistema como anteriormente:

```
node index.js
```

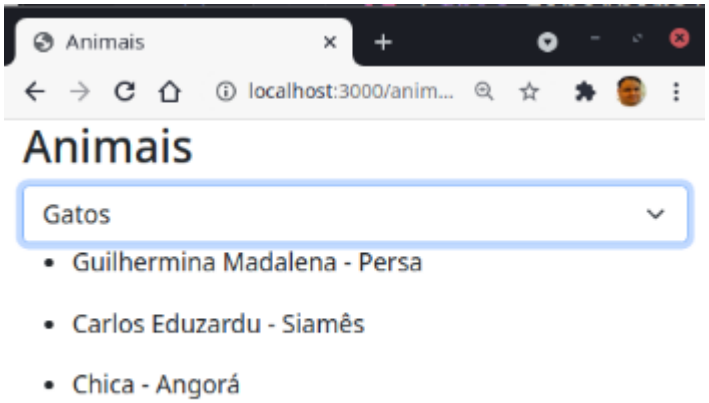
Acesse no navegador:



Legenda da imagem



Legenda da imagem



Legenda da imagem

Mesmo comportamento anterior, porém agora com reatividade e usando o Vue.js

Conclusão

Usamos o Vue.js para mostrar o conceito de reatividade no nosso exemplo da aula anterior. Ficamos com um código javascript bem menor, de melhor manutenção e com mais possibilidades de crescimento com produtividade.

O Vue.js é uma ferramenta com muito mais funcionalidades que as vistas aqui, mas já dá deu pra ter uma ideia de como ele funciona gerenciando os dados e como ela possibilita o desenvolvimento de aplicações reativas.

Você irá nessa disciplina conhecer outras bibliotecas com as mesmas funcionalidades, porém tratadas de forma um pouco diferente. Até lá estude mais o Vue.js em: https://vuejs.org/