Plataformas de aplicações Web

Aula 07 - Comportamento dinâmico com Javascript

Material Didático do Instituto Metrópole Digital - IMD

Apresentação

Sim, é comum ferramentas como o Bootstrap ou o Materialize serem categorizadas como ferramentas front-end, e realmente são, mas aqui elas estão categorizadas como Bibliotecas de Componentes UI o que não as tornam menos importantes.

Nessa disciplina estamos definindo Plataformas front-end reativas as tecnologias que facilitam a criação de páginas com componentes dinâmicos, fornecendo funcionalidades que permitem que elementos na página sejam adicionados, removidos ou alterados sem a necessidade que uma nova página seja gerada no lado do servidor.

Vamos nessa aula ver o as diferenças entre comportamento dinâmico e reativo, vendo inicialmente um exemplo dinâmico em Javascript puro, acessando dados JSON e alterando elementos de do HTML que ele está implementado.

Comportamento reativo

Imagine que você tenha uma página web que lista animais que está em sua base de dados e você deseja criar uma funcionalidade onde o usuário pode escolher listar somentes cães, ou gatos e nesse instante você gostaria que a lista de animais seja alterada, respeitando a opção do usuário, mas que não deseja que a página seja completamente carregada novamente, mas somente a lista em si seja alterada dinamicamente.

O que está descrito acima é um comportamento "dinâmico" de uma página, mas não necessariamente "reativo".

A distinção entre comportamento dinâmico e reativo precisa inicialmente que tenhamos o seguinte em mente:

- Todo comportamento reativo é dinâmico
- Nem todo comportamento dinâmico é reativo

Dinâmico significa somente que a página realizará operações que alteram os seus elementos usando funções que rodam no cliente. Isso é o suficiente para se fazer qualquer tipo de página interativa.

O comportamento reativo significa que o DOM e os elementos devem ser exibidos reagindo automaticamente aos dados presentes em variáveis do Javascript. Por exemplo, você altera um Array de animais e deve existir funcionalidades que a lista de animais automaticamente mude pare refletir esse novo array, sem você precisar manualmente remover os elementos da lista HTML e adicionar novos.

O comportamento reativo real é complicado (porém possível) de ser implementado

em Javascript puro pois a linguagem não oferece reatividade de forma nativa, sendo necessária a criação de camadas extras para que isso seja possível.

Existem bibliotecas como o React, VueJS e Svelte que podem te ajudar e vamos conhecê-las melhor em aulas seguintes.

Comportamento dinâmico com Javascript

Vamos fazer um pequeno projeto para exemplificar um comportamento dinâmico de uma página utilizando Javascript puro no lado do cliente, sem nenhuma biblioteca.

Para esse projeto iremos precisar criar um sistema em Express um pouquinho mais simples que o comum e que irá ter as seguintes funcionalidades:

- Uma rota "/animais" que irá retornar um arquivo HTML estático que está em "/pages/animais.html"
    - Esse arquivo terá HTML + CSS + Javascript com um SELECT com as opções de "Cães" ou "Gatos" para se obter a lista de animais.
    - Uma lista (UL) com a relação de animais que foi retornada de forma assíncrona, por uma consulta a uma API (que estará no mesmo projeto)
    - A ação de reagir a mudança de opção no select (Cães ou Gatos) será tratada por um Javascript na mesma página que irá obter a nova lista de animais em JSON consultando a rota correta (/dados/caes.json ou /dados/gatos.json). Essas rotas são arquivos JSON estáticos servidos pelo mesmo sistema Express que está servindo essa página. Não existirá banco de dados relacional, somente arquivos JSON estáticos, por enquanto.
- Uma rota para servir arquivos estáticos dentro da pasta "dados". Essa rota será "/dados" e automaticamente servirá qualquer arquivo dentro da pasta dados (no caso será caes.json e gatos.json)
- Duas funções Javascript no animais.html
    - escolherAnimal() será chamada quando um novo tipo de animal for escolhido no select (Cães ou Gatos). Ela irá usar a função "fetch" para consultar ou "/dados/caes.json" ou "/dados/gatos.json" para obter a lista de animais.
    - listarAnimais(animais) recebe uma lista de animais como parâmetro e irá realizar duas operações:
    - Limpar os animais atuais na lista (UL) de animais da página
    - Adicionar um a um os novos elementos LI na lista com o nome - raça de cada animal recebido como parâmetro

Criando o projeto - Dados JSON

Vamos criar uma pasta chamada "purejavascript" para o nosso projeto em Javascript puro, criar os documentos HTML e JSON necessários e criar um aplicativo Express para servir esses documentos para o navegador.

Para criar o projeto inicialmente faça:

```
mkdir purejavascript
cd purejavascript
npm init -y
npm install express
```

Agora vamos criar uma pasta chamada "pages" e uma chamada "dados"

```
mkdir pages
mkdir dados
```

Dentro da pasta "dados" crie dois arquivos chamados "caes.json" e "gatos.json" com os seguitnes conteúdos:

caes.json:

```
[
  {
    "nome": "Bruce Wayne",
    "raca": "Pug"
    },
  {
    "nome": "Firmino Joaquino",
    "raca": "Labrador"
    },
  {
    "nome": "Odisnéia Filandra",
    "raca": "Pinscher"
  }
]
```

gatos.json:

```
[
  {
```

```
    "nome": "Guilhermina Madalena",
    "raca": "Persa"
  },
  {
    "nome": "Carlos Eduzardu",
    "raca": "Siamês"
  },
  {
    "nome": "Chica",
    "raca": "Angorá"
  }
]
```

Esses arquivos são no formato JSON e cada um tem um Array com 3 objetos cada. Cada um desses objetos são um cão ou um gato (dependendo do arquivo) com os atributos "nome" e "raca". Evitamos usar acentos e caracteres especiais nos nomes das chaves dos atributos, mas no conteúdo deles é possível sim. Isso é uma prática comum para evitar incompatibilidades na conversão de JSON para objetos Javascript.

Criando o projeto - HTML+CSS+Javascript

Dentro da pasta "pages" crie um arquivo chamado "animais.html" com o seguinte conteúdo:

```
<!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>
  <title>Animais</title>
  <script>
    async function escolherTipoAnimal() {
      // obtém o valor do tipo de animal escolhido ('caes' ou
'gatos')
      const tipoAnimal =
document.getElementById('tipoAnimal').value;

      if (tipoAnimal == '') return; // finaliza se nenhum tipo foi
escohido

      // cria uma URL com o caminho correto para o tipo escolhido
      // 'dados/caes.json' ou 'dados/gatos.json'
      const animaisJSONURL = `dados/${tipoAnimal}.json`;

      // usa o fetch para obter o conteúdo do documento JSON da url
gerada
      const resultado = await fetch(animaisJSONURL);

      if (resultado.ok) {
        // Caso a requisição do documento tenha sido realizada com
sucesso
        // coloca o conteúdo json do documento (um array de animais)
na variável "animais".
        const animais = await resultado.json();

        // chama a função listarAnimais com o array recebido do
documento json.
        listarAnimais(animais);
      }
    }

    async function listarAnimais(animais) {
      // obtém o elemento UL com a lista de animais
      const animaisUL = document.getElementById('animaisUL');

      // limpa todo os LI da lista de animais (limpa a lista UL)
      animaisUL.innerHTML = "";

      // para cada animal do array passado
```

```
      for (const animal of animais) {
        // cria um novo elemento LI,
        const novoLI = document.createElement('LI');

        // adiciona o nome - raça do animal como conteúdo do LI
        novoLI.innerHTML = `${animal.nome} - ${animal.raca}`;

        // e adiciona o LI como novo filho da lista de animais
        animaisUL.appendChild(novoLI);
      }
    }
  </script>
</head>
<body>
  <div class="container">
    <h1>Animais</h1>
    <select id="tipoAnimal" onChange="escolherTipoAnimal()"
class="form-select">
      <option value="">Escolha um tipo abaixo</option>
      <option value="caes">Cães</option>
      <option value="gatos">Gatos</option>
    </select>

    <ul id="animaisUL">
    </ul>

  </div>
</body>
</html>
```

Repare que nesse HTML estamos usando o Bootstrap somente para estilizar melhor os nossos elementos.

O animais.html contém um SELECT com o id="tipoAnimal" com as opções com value "caes" e "gatos". No onChange="escolherTipoAnimal()" o HTML está sendo informado que quando o usuário mudar a opção uma função chamada escolherTipoAnimal() será executada em Javascript.

A função escolherTipoAnimal() realiza algumas operações.

Primeiro ela obtém o valor do tipo de animal escolhido ('caes' ou 'gatos') do select. Caso não seja nada (o usuário escolheu a opção em branco chamada "Escolha um tipo abaixo" ela simplesmente retorna sem fazer nada.

```
const tipoAnimal = document.getElementById('tipoAnimal').value;
if (tipoAnimal == '') return;
```

Em seguida é criada uma URL específica para a opção escolhida com o código:

```
const animaisJSONURL = dados/${tipoAnimal}.json`;
```

Esse código atribui a variável animaisJSONURL os valores possíveis: 'dados/caes.json' ou 'dados/gatos.json', de acrodo com o que foi escolhido.

Em seguida é utilizada a função fetch para acessar essa URL e se obter o conteúdo JSON do documento escolhido:

```
const resultado = await fetch(animaisJSONURL);
```

O fetch retorna um Promise, e é assíncrono, portanto você pode utiliza a sintaxe "await" para se "aguardar" o retorno dele sem bloquear a interface do usuário. Para mais informaçòes sobre o await visite: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await

Em um caso real, com o sistema em produção na Internet, o retorno do "fetch(animaisJSONURL)" pode falhar, caso a internet esteja fora do ar, por exemplo, ou algum erro aconteca no servidor que está fornecendo os dados JSON. Então é uma boa prática verificar se o retorno foi obtido com sucesso (verificando o "resultado.ok") e só então obter o conteúdo JSON desse retorno (conteúdo do documento que foi retornado):

```
if (resultado.ok) {
  // Caso a requisição do documento tenha sido realizada com sucesso
  // coloca o conteúdo json do documento (um array de animais) na
variável "animais".
  const animais = await resultado.json();

  // chama a função listarAnimais com o array recebido do documento
json.
  listarAnimais(animais);
}
```

Repare que se o resultado estiver "ok" é obtido na variável "animais" o conteúdo do documento JSON que foi solicitado. Então é chamada a função listarAnimais(animais) para exibir esse Array JSON como na lista no HTML, trocando seu conteúdo por um novo.

A função listarAnimais(animais) realiza a troca do conteúdo da lista UL de animais por um novo com uma sequência de operações que estão comentadas no código HTML acima. Basicamente é obtido o elemento UL, seu conteúdo é limpo atribuindo o innterHTML = "", depois é feito uma iteração na lista de animais recebidas e para cada um é criado em Javascript um elemento LI e adicionado esse elemento na lista de animals usando o método appendChild.

Temos agora todos os documentos necessários no nosso projeto, com exceção da aplicação Express que vai servir todos eles para o navegador nas rotas que desejamos.

Criando o projeto - Express app

Como já dito, precisamos de um sistema para servir os documentos que criamos nas rotas planejadas. Isso será feito com um sistema Express. Para isso, vamos criar um arquivo chamado "index.js" e colocar o seguinte código:

```
const express = require('express');
const path = require('path');
const app = express();

app.get('/animais', (req, res) => {
  res.sendFile(path.join(\_\_dirname, './pages/animais.html'));
});

app.use('/dados', express.static('dados'));

app.listen(3000, () => {
  console.log('Servidor rodando na porta 3000');
});
```

Repare que o trecho abaixo informa ao Express que a rota "/animais" retorna irá retornar o conteúdo do arquivo animais.html que está na pasta "pages" do projeto (que já foi criada):

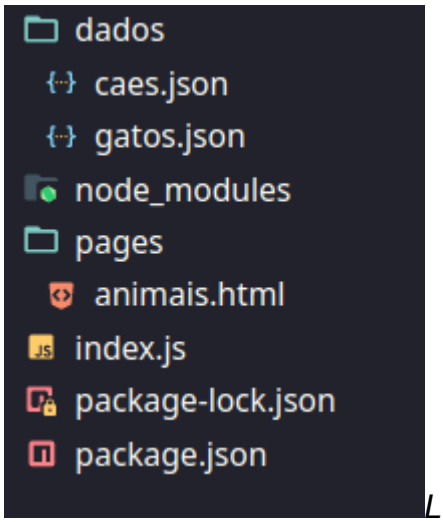
```
app.get('/animais', (req, res) => {
  res.sendFile(path.join(\_\_dirname, './pages/animais.html'));
});
```

O seguinte trecho diz que a pasta "dados" será servida na rota "/dados" e todos os documentos dentro de dados estarão disponíveis como arquivos estáticos:

```
app.use('/dados', express.static('dados'));
```

Essa é uma forma rápida de se usar o Middleware "static" do Express para servir arquivos em pastas de maneira simples e direta. No nosso caso os arquivos que estão dentro da pasta "dados" são os documentos JSON: caes.json e gatos.json.

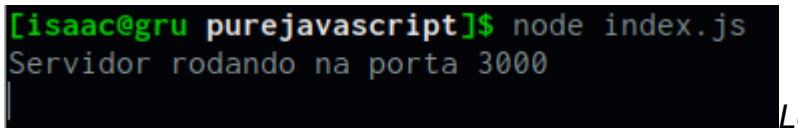
O projeto final ficou com a seguinte estrutura de pasta:



Legenda da imagem

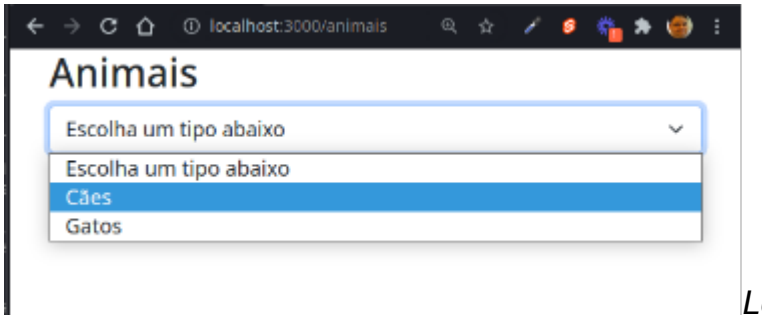
Pronto, vamos rodar o nosso projeto. Na pasta do projeto execute o index.js

```
node index.js
```



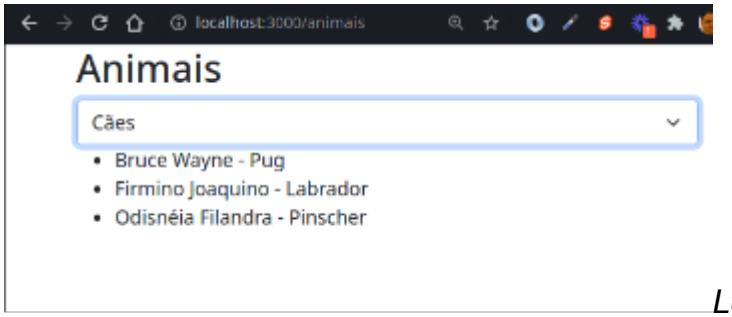
Legenda da imagem

Como configurado, ele estará rodando na porta 3000 e pode ser acessado pelo navegador em http://localhost:3000/



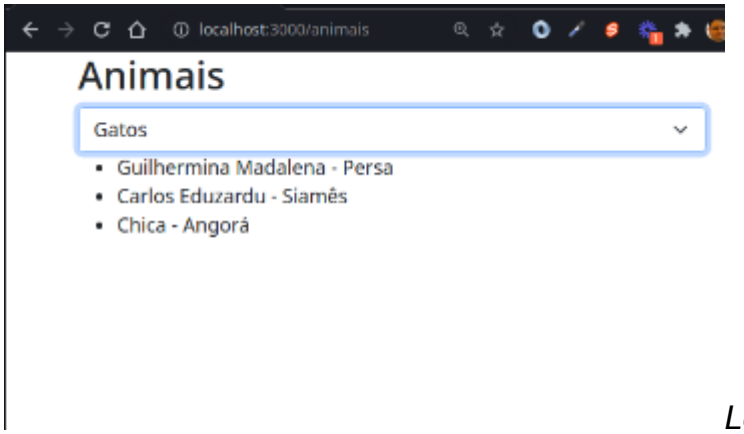
Legenda da imagem

Exibindo as opções



Legenda da imagem

Escolhendo "Caẽs"



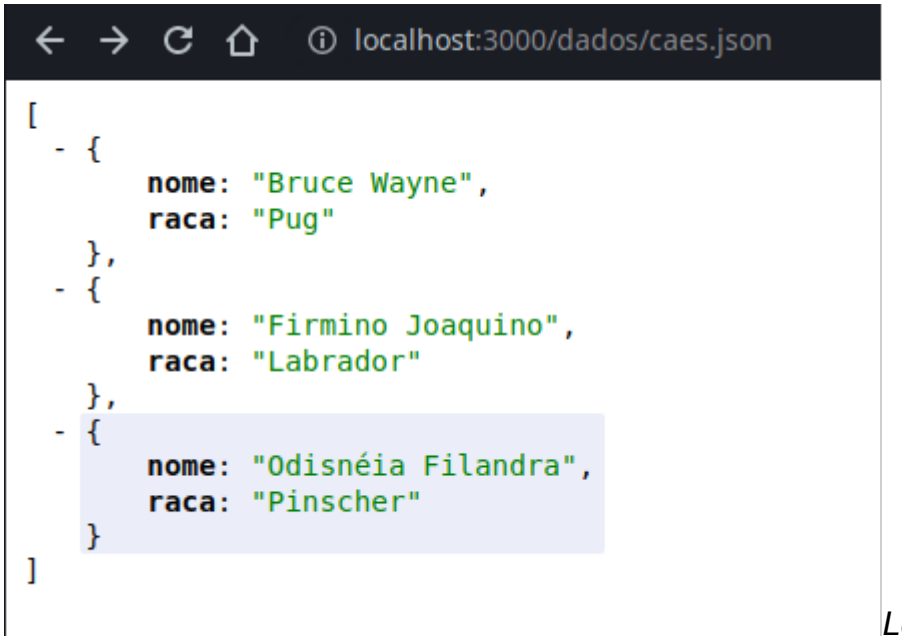
Legenda da imagem

Escolhendo "Gatos"

Só por curiosidade, vamos tentar acessar diretamente os dados JSON de Cães e Gatos, sem usar o HTML. Pra isso acesse pelo navegador a URL http://localhost:3000/dados/caes.json e http://localhost:3000/dados/gatos.json

Legenda da imagem

Dados JSON de Gatos



Legenda da imagem

Dados JSON de Cães

Curiosidade: Seu navegador pode não exibir os dados JSON coloridos e formatados dessa forma. Para isso funcionar você pode instalar uma extensão chamada JSONView disponível para o Google Chrome. Existem outras extensões que exibem JSON de forma mais formatada em outros navegadores também, se preferir.

Conclusão

Vimos como criar um sistema simples em HTML, Bootstrap e Javascript para acessar dinamicamente dados JSON na internet e dinamicamente alterar o conteúdo de uma lista que está em nosso documento HTML sem precisar recarregar toda a página.

Esse comportamento não é um comporta exatamente chamado de "reativo", pois a mudança do estado do HTML foi realizada por comandos específicos em Javascript que manipulou o DOM. Vamos nas próximas aulas conhecer algumas bibliotecas que ajudam esse processo tornando-o mais simples e efetivamente reativo.