Innominatum CONSULTORIA

Programação em JavaScript

ServiceNow - Deloitte

Rodrigo Costa

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

Innominatum CONSULTORIA

JavaScript

Módulo 2

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

Innominatum CONSULTORIA

Storage & Modules

Sessão 10

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

Objetivo geral

Capacitar os formandos na construção de aplicações JavaScript organizadas e persistentes, introduzindo mecanismos de armazenamento local para manutenção de estado entre sessões e promovendo a divisão estruturada do código através de módulos ES6. A sessão foca na transição de scripts monolíticos para uma arquitetura modular e sustentável, desenvolvendo a capacidade de organizar responsabilidades em ficheiros distintos e de persistir dados no navegador de forma controlada e padronizada.

Objetivos específicos

Ao final da sessão, os formandos deverão ser capazes de:

- Distinguir **localStorage** de **sessionStorage**, compreendendo seus ciclos de vida e casos de uso;
- Persistir objetos JavaScript no storage utilizando **JSON.stringify()** e **JSON.parse()**;
- Aplicar ES6 Modules utilizando **import** e **export** (named e default);
- Organizar código em múltiplos ficheiros, separando responsabilidades (ex: api.js, dom.js, utils.js);
- Distinguir **CommonJS** de **ESModules**, compreendendo por que **ESModules** é o padrão moderno e alinhado com Angular.

Innominatum CONSULTORIA

Web Storage API

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

O problema

Até agora, trabalhamos com o que chamamos de “Estado volátil”. Sempre que atualizamos a página:

- **O estado desaparece**: O motor JS reinicia do zero;
- **Arrays voltam ao valor inicial**: Todas as variáveis declaradas no código são limpas da memória;
- **Dados perdem-se**: Não há memória entre sessões de navegação.

É óbvio dizer que as aplicações reais não funcionam assim.

O que é web storage?

A **web storage API** é um conjunto de ferramentas nativas do navegador que nos permite:

- **Guardar dados no navegador**: Armazenar informações diretamente no dispositivo do utilizador, sem depender de um servidor externo para cada pequena ação;
- **Persistir informação localmente** Decidir se os dados devem ser temporários ou permanentes;
- **Manter estado entre reloads**: Garantir que as escolhas como “Dark Mode” ou filtros de produtos sobrevivam ao carregar F5;

Ela é uma API simples e está integrada ao navegador.

Dois tipos de storage

Existem dois tipos principais:

Tipo	Duração
localStorage	Persiste até ser removido
sessionStorage	Persiste até fechar o separador

A diferença entre os tipos de armazenamento está no tempo de vida.

localStorage

LocalStorage é a memória de longo prazo do navegador, as suas principais características são:

- **Persistir após reload:** Ao contrário das variáveis de JS, que os dados são limpos quando a página é atualizada;
- **Persiste após fechar o navegador**: Os dados permanecem gravados no disco rídigo do utilizador mesmo que ele desligue o dispositivo. Só desaparecem se foram apagados via código pelo utilizador ou com a limpeza do cache ou dados de navegação do navegador;
- **Armazenamento por domínio**: Os dados guardados por um site, estão seguros e isolados, não podendo serem lidos por outro site, isso garante a **privacidade e integridade** das informações.

localStorage: Armazenamento e leitura

```
// localStorage
localStorage.setItem("nome", "Ana");
//2. Estamos armazenando no localStorage do navegador uma chave e valor;
```

```
const nome = localStorage.getItem("nome");
// 5. recupera o valor da chave "nome" do localStorage e a armazena na variável nome.
```

Ao ler o conteúdo a resposta sempre será uma string ou null.

Remover dados

Os atributos removeItem() e clear() removem os dados do navegador.

```
localStorage.setItem("apelido", "Silva");
// 18. Estamos armazenando no localStorage
const apelido = localStorage.getItem("apelido");
// 19. Recupera o valor da chave "apelido"
console.log(apelido)
```

```
localStorage.removeItem("apelido");
// 29. Remove a chave "apelido" do localStorage
localStorage.clear();
// 31. Limpa todo o localStorage,
// removendo todas as chaves e valores armazenados.
```

sessionStorage

```
// sessionStorage

sessionStorage.setItem("tema", "light")
// 36. Armazena a chave "tema" com o valor "light" no sessionStorage.
let tema = sessionStorage.getItem("tema");
// 37. Recupera o valor da chave "tema" do sessionStorage e a armazena na variável tema.
console.log(tema)

const isDark = tema === "dark" ? body.classList.add('body') : body.classList.remove('body');
// 42. Verifica se o valor de tema é "dark". Se for, adiciona a classe "body" ao elemento body,
// caso contrário, remove a classe "body".
```

Tem a mesma funcionalidade do localStorage, contudo a informação desaparece quando o separador é fechado.

Quando usar cada um?

Use **localStorage** quando:

- **Precisa manter preferências**: Configurações que o utilizador não quer repetir (ex: dark mode, idioma, tamanho do texto);
- **Precisa manter login simples**: Armazenar o nome de utilizador ou um token de acesso para que ele não tenha de fazer o login sempre que abre o navegador;
- **Precisa manter estado duradouro**: Dados que devem sobreviver a dias ou semanas;

Use **sessionStorage** quando:

- **Dados temporários**: Informações que só fazem sentido naquela navegação específica (ex: o termo que o utilizador acabou de pesquisar);
- **Informação sensível de sessão**: Dados que devem ser destruídos assim que o utilizador fechar a aba por questões de privacidade ou segurança;
- **Processos de checkout/formulários**: Se o utilizador está num formulário de 3 passos, os dados do passo 1 e 2 podem ficar aqui para caso o utilizador volte atrás nas etapas os dados ainda estejam preenchidos.

Innominatum CONSULTORIA

Persistência de Estado com JSON

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

O problema real

Storage guarda apenas **strings**, mas as aplicações trabalham com dados primitivos, como **strings**, **números**, e **booleanos**, além dos dados complexos como **arrays**, e **objetos**. Então precisamos utilizar uma outra forma de guardamos essa informação dentro do storage.

A solução para os dados complexos

A solução seria a **serialização**, vamos converter os objetos em texto.

```
const tarefa = {
  texto: "Estudar módulos",
  concluida: false
};

localStorage.setItem("tarefa", JSON.stringify(tarefa));
// 2. Armazena o objeto tarefa no localStorage,
//  convertendo-o para uma string JSON usando JSON.stringify().

console.log(tarefa);
```

```
Tarefa:                                                exemplo-02.js:10
▼ {texto: 'Estudar módulos', concluida: false} i
    concluida: false
    texto: "Estudar módulos"
```

Recuperar objeto do storage

```
// Caputar o objeto do localStorage
const tarefaString = localStorage.getItem("tarefa");
const tarefaObjeto = JSON.parse(tarefaString);
```

Persistindo arrays

A lógica será a mesma, ao persistir o dado no localStorage, o array ganhará uma chave, e será convertido em uma String JSON.

```
// Persistindo com Arrays

const tarefas = [
  {texto: "JS", concluida: false},
  {texto: "Storage", concluida: true}
];

localStorage.setItem("tarefas", JSON.stringify(tarefas));
// 25. Convertemos o Array para uma String JSON
```

Recuperar lista persistida

```
const recuperandoTarefas = JSON.parse(localStorage.getItem("tarefas")) || [];
// 28. estamos recuperando as tarefas, ou se elas estiverem vazias,
// retorna um array vazio.

console.log('Tarefas recuperadas:', recuperandoTarefas);
// 32. Exibimos as tarefas recuperadas no console, mostrando o array de objetos.
console.log('Recuperando tarefas:', recuperandoTarefas[0].texto);
// 34. Exibimos o texto da primeira tarefa recuperada,
// acessando a propriedade "texto" do primeiro objeto no array recuperado.
```

ES6 Modules (import/export)

O problema do ficheiro único

Quando o projeto cresce:

- O ficheiro fica gigante e com isso:
  - Dificulta a manutenção;
  - As responsabilidades ficam misturadas;
  - Risco de conflito de variáveis.

A solução é modularizar o sistema para dividir responsabilidades.

O que são módulos?

Um módulo é:

- Um ficheiro JS:
  - Com escopo próprio;
  - Que pode exportar funcionalidades;
  - E importar de outros ficheiros;

Quando trabalhamos com módulos, cada ficheiro torna-se uma unidade lógica.

Como ativar módulos no HTML

Para ativar o módulo no HTML será necessário adicionar uma propriedade type, passando o valor module para ela, dentro da tag de script. Sem o type="module", o import não terá funcionalidade.

```html
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <script type="module" src="script.js"></script>
    <title>Módulos</title>
</head>
<body>
    <h1>Trabalhando com módulos</h1>
</body>
</html>
```

Named export e Import named

Para utilizarmos a exportação nomeada, precisamos inserir a palavra reservada export na frente da declaração da função. Na sequência importamos a função somar do módulo utils.

```
// utils.js
// Exportação nomeada da função somar
export function somar(a,b) {
    return a + b;
}
```

```
import { somar } from "../utils/utils.js";
// Importando a função somar do arquivo utils.js
console.log("Resultado da Soma: ", somar(2, 3))
```

Default Export e Import Default

No default export e import default a exportação da função é nomeada com a palavra reservada default, o que permite que na importação, seja utilizado um nome personalizado.

```
// Exportação default da função buscarDados
export default function buscarDados() {
    console.log("Buscando dados...")
}
```

```
import { somar } from "../utils/utils.js";
// 2. Importando a função somar do arquivo utils.js
import buscarDados from "../services/buscarDados.js"
// 3. Importação default do serviço buscar dados

console.log("Resultado da Soma: ", somar(2, 3));
buscarDados();
```

Named vs Default

Named Export	Default Export
Pode ter vários	Apenas um
Usa {}	Não usa {}
Nome fixo	Nome livre

Innominatum CONSULTORIA

Modularização: CommonJS vs ESModules

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

CommonJs (Node.js)

```
const somar = require("../utils/sum.js")

const resultado = somar(2,5);

console.log("Resultado da Soma: ", resultado)
```

```
function somar(a, b) {
    return a + b;
}

module.exports = somar;
```

Possui limitações por não ser nativo no navegador, é baseado em require, a execução é síncrona e é menos otimizado para bundlers modernos.

ESModules (ESM)

Usamos o ESModules, porque:

- É padrão oficial da linguagem;
- Funciona nativamente no navegador;
- É usado pelo Angular (framework da nossa formação);
- É suportado por bundlers modernos;
- Permite análise estática de dependências.

```
// ESModules (ESM)

export function somar(a, b) {
    return a + b;
}
```

```
import { somar } from "../utils-es/utils-es.js"

const resultado = somar(20,30);

console.log("Resultado ESModule: ", resultado);
```

Diferenças principais

CommonJS	ESModules
Require()	Import
Module.exports	Export
Node tradicional	Navegador + Node moderno
Execução síncrona	Baseado em módulos estáticos

Síntese

- **Persistência de Estado com Web Storage API**: Fazendo a distinção entre **localStorage** de **sessionStorage** com base no ciclo de vida dos dados;
- **Serialização e desserialização com JSON**: Aplicação de **JSON.stringify**() e **JSON.parse**(), para converter objetos JS em texto armazenável;
- **Modularização com ES6 Modules**: Introdução ao padrão moderno de organização de código através de import e export;
- **Contextualização de CommonJS vs ESModules**: Entendimento da evolução do sistema de módulos em JS.

Conclusão

Innominatum CONSULTORIA

Programação em JavaScript

ServiceNow - Deloitte

Rodrigo Costa

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL