Integração Básica de APIs e Local Storage

O que já vimos?

- Como criar aplicações React lidando com interatividade do usuário
- O que é e como funciona uma API
- Como consumir uma API no Postman

O que vamos ver hoje?

- LocalStorage
- Como consumir uma API usando Javascript
- Como lidar com dados externos no React
- Dados dinâmicos nas nossas aplicações

Local Storage

Motivação 🗄️

- Até agora, só lidamos com informações criadas **durante a vida da aplicação**
- Ao atualizar a página, **perdemos** todos os dados!
- Não conseguimos **persistir** dados
- Uma possível solução para isso é o **Local Storage**

Local Storage 🗄️

- É uma funcionalidade **do navegador** que permite que dados sejam **salvos** e **lidos**

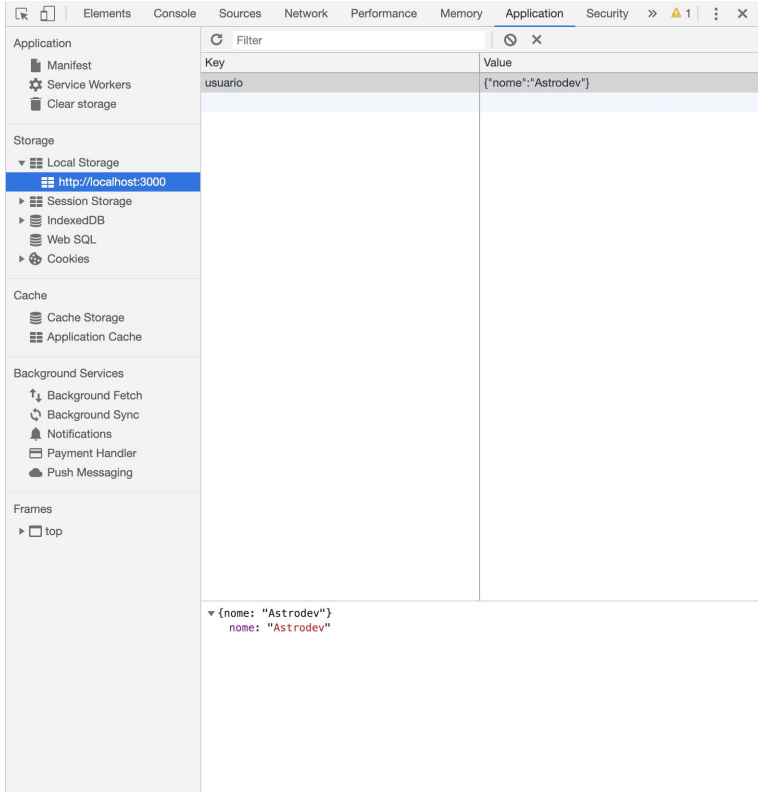
- Dados são armazenados associados a um **domínio** (URL) e ficam guardados mesmo que o navegador seja fechado

- Está disponível globalmente no navegador por meio do objeto **localStorage**

Vendo os dados salvos 

- Na aba **Application** do DevTools, temos acesso aos dados salvos no localStorage

- Usamos isso para ver o funcionamento do código



Guardando dados 🗄️

- Para guardar dados, usamos a função `setItem()`

- Ela recebe dois parâmetros:
  - **Chave:** identificador do que guardaremos
  - **Dados:** dados a guardar. Devem ser uma string

```
localStorage.setItem("usuario", "Fravo")
```

Buscando dados 🗄️

- Para buscar dados, usamos a função `getItem()`

- Ela recebe um parâmetro:
  - **Chave:** identificador do que estamos buscando. Deve ser **a mesma usada para guardar** o dado

```
localStorage.getItem("usuario")
```

Problema 

- Local Storage **só armazena strings**

- Mas frequentemente vamos querer guardar outros tipos de valores, como **arrays** e **objetos**

- Para isso, podemos **transformá-los** em string na hora de salvar e voltar ao formato original na hora de pegá-los de volta

Solução 🗄️

- **Array/Objeto ⇒ String**
  - `JSON.stringify()` transforma objetos e arrays em string

- **String ⇒ Array/Objeto**
  - `JSON.parse()` transforma string em objetos e arrays

Como usar Local Storage

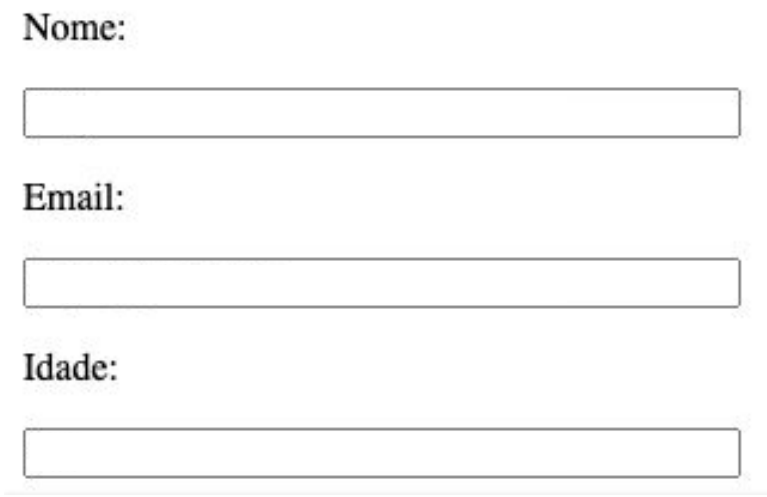
```
const novoUsuario = { nome: "Flávio", idade: 22 };

localStorage.setItem("usuario", JSON.stringify(novoUsuario));

const usuarioToString = localStorage.getItem("usuario");

const usuarioObjeto = JSON.parse(usuarioToString);

console.log(usuarioObjeto); // { nome: "Flávio", idade: 22 }
console.log(usuarioObjeto.nome); // Flávio
```

Exercício 1

- Crie um componente de formulário com 3 inputs controlados

- Esses inputs representam a **edição do perfil** de um usuário e devem ser: **nome**, **email** e **idade**

Vamos ver na prática! 🔬



Exercício 2

Crie dois botões no seu formulário:

- Um para **salvar** os dados no localStorage
- Um para **pegar** os dados salvos e mostrar nos inputs

Exemplo do Formulário 🗄️

- Esse comportamento não é o melhor para um formulário de perfil!

- Algo mais interessante seria eliminar os botões:
  - Salvar os dados **automaticamente** conforme eles são escritos
  - Assim que a tela abrir, **automaticamente** o formulário é preenchido com os dados corretos

Requisições em Javascript

axios 🔧

- **Biblioteca** para fazer requisições
- "Igual" ao Postman, só que dentro do código
- Existem formas nativas (sem bibliotecas), mas o *axios* facilita várias coisas para nós
- Instalando:

```
$ npm install axios
```

axios 🔧

- Sintaxe - Exemplo

```
axios.get("https://pokeapi.co/api/v2/pokemon"), {
    headers: {
        "Content-Type": "application/json",
        "Authorization": "Bearer " + localStorage.getItem("token")
    }
}
```

Problemas 👻

- E se a requisição demorar muito?
  - Devemos parar a execução do código? NÃO

- E se der erro na requisição?
  - Parâmetros errados
  - Servidor com problema
  - Usuário sem internet

Tempo de Requisição 🕐

- Não temos controle sobre o tempo da requisição
- Não queremos que a aplicação fiquei travada enquanto esperamos
- Javascript criou uma solução: **assincronicidade**

Sincronicidade 🕐

- Normalmente, o código é executado linha após linha
- Quando uma função é chamada, o código aguarda a execução dela para prosseguir

Assincronicidade 🕐

- Podemos fazer com que o Javascript execute funções demoradas de forma assíncrona

- A diferença é que o código não espera sua conclusão para prosseguir

- Nova estrutura nos permite lidar com isso de forma mais intuitiva: **Promises**

Promises 🕐

```
const request = axios.get("https://pokeapi.co/api/v2/pokemon",{
  headers: {
    "Content-Type": "application/json",
    "Authorization": "Bearer " + localStorage.getItem("token")
  }
})

request.then((response)=> {
  console.log(response.data)
})
```

Tratando erros 🤷

- Até agora, trabalhamos somente com dados e códigos criados por nós
- Todos os erros eram **responsabilidade nossa**
- Quando usamos integrações externas, estamos sujeitos a **erros que fogem do nosso controle**
- Precisamos **tratar** esse erro para que nossa aplicação não pare de funcionar

Tratando erros em promises 🤷

- Depois de todos os `.then()`, colocamos um `.catch()`

```
const request = axios.get("https://pokeapi.co/api/v2/pokemon",{
  headers: {
    "Content-Type": "application/json",
    "Authorization": "Bearer " + localStorage.getItem("token")
  }
})

request.then((response)=> {
  console.log(response.data)
}).catch((error)=> {
  console.log(error.message)
})
```

axios - Sintaxe

Sem body

```
const request = axios.get("https://pokeapi.co/api/v2/pokemon",{
  headers: {
    "Content-Type": "application/json",
    "Authorization": "Bearer " + localStorage.getItem("token")
  }
})
request.then((response)=> {
  console.log(response.data)
}).catch((error)=> {
  console.log(error.message)
})
```

Com body

```
const body = {
  "username": "admin",
  "password": "[PASSWORD]"
}
const request = axios.post("http://localhost:8080/login",body,{
  headers: {
    "Content-Type": "application/json"
  }
})
request.then((response)=> {
  console.log(response.data.token)
  localStorage.setItem("token", response.data.token)
}).catch((error)=> {
  console.log(error.message)
})
```

axios - Sintaxe

Sem body

```
const request = axios.get("https://pokeapi.co/api/v2/pokemon",{
  headers: {
    "Content-Type": "application/json",
    "Authorization": "Bearer " + localStorage.getItem("token")
  }
})
request.then((response)=> {
  console.log(response.data)
}).catch((error)=> {
  console.log(error.message)
})
```

Com body

```
const body = {
  "username": "admin",
  "password": "[PASSWORD]"
}
const request = axios.post("http://localhost:8080/login",body,{
  headers: {
    "Content-Type": "application/json"
  }
})
request.then((response)=> {
  console.log(response.data.token)
  localStorage.setItem("token", response.data.token)
}).catch((error)=> {
  console.log(error.message)
})
```

Requisições no React

Buscando Dados 

- Normalmente queremos buscar dados para mostrar na tela ou tomar alguma ação baseada neles

- Depois de pegar o dado, é necessário guardá-lo em algum lugar

- Para isso, usamos o **estado**

Buscando Dados Automaticamente 🎲

- Frequentemente, vamos querer carregar os dados automaticamente assim que a tela carrega
- Requisições para buscar dados devem ser executadas **depois** da renderização dos componentes
- Não queremos que a tela fique travada enquanto a requisição é feita
- Para isso, usamos useEffect

Fazendo a Requisição 🎲

- Para buscar dados automaticamente, devemos usar o método de lifecycle `useEffect()`

- Fazer a requisição usando o axios e colocar o resultado no **estado**

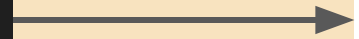
Enviando Dados 🎲

- Requisições para enviar dados normalmente ocorrerão com base em um **evento** (ação do usuário)

- Frequentemente, mandaremos dados vindos de um **formulário**

- Para fazer isso, usamos os métodos que lidam com eventos

```
function App() {
  const [pokemon, setPokemon] = useState([])
  const pegarPokemons = () => {
    axios.get("https://pokeapi.co/api/v2/pokemon", {
      headers: {
        "Content-Type": "application/json"
      }
    })
      .then((response) => {
        console.log(response.data.results)
        setPokemon(response.data.results)
      })
      .catch((error) => {
        console.log(error.message)
      })
  }
  useEffect(() => {
    pegarPokemons()
  }, [])
  const renderedPokemons = pokemon.map((pokemon) => {
    return (
      <p key={pokemon.name}>{pokemon.name}</p>
    )
  })
  return (
    <>
      {renderedPokemons}
    </>
  )
}
```



bulbasaur
ivysaur
venusaur
charmander
charmeleon
charizard
squirtle
wartortle
blastoise
caterpie
metapod
butterfree
weedle
kakuna
beedrill
pidgey
pidgeotto
pidgeot
rattata
raticate