

# FETCH & ASYNC/AWAIT



DEVinHouse



Parcerias para desenvolver a sua carreira



**SENAI**

<LAB365>

- Revisão
  - **Promise.then(f).catch(g).finally(h)**
- Fetch
- Async Await
- Try Catch

- Uma promise sempre estará em algum destes estados:
  - ***pending*** (pendente): 🕒  
Estado inicial, que não foi realizada nem rejeitada
  - ***fulfilled*** (realizada): ✅  
A execução foi concluída com sucesso
  - ***rejected*** (rejeitado): ❌  
Ocorreu alguma falha na execução

- Promise
  - `new Promise((resolve, reject) => { resolve('a'); })`  
`.then(result => {}).catch(error => {}).finally(() => {});`
- Resolve
  - Função 1º parâmetro de promise, que ao executar conclui a promise 
  - `resolve('valor enviado para o then');`
- Reject
  - Função 2º parâmetro de promise, que ao executar termina a promise 
  - `reject('valor enviado para o catch');`

- Then
  - Método de uma promise que permite a definição de uma função a ser executada quando a promise se resolver com sucesso 
  - **.then**(paramDeResolve => {});
- Catch
  - Método que permite a definição de uma função a ser executada quando a promise não se resolver com sucesso 
  - **.catch**(paramDeReject => {});
- Finally (sempre executa, independente de sucesso ou falha)
  - **.finally**(( ) => {});

# FETCH



- A Fetch API fornece uma interface para buscar recursos (por exemplo, em toda a rede). Parecerá familiar para qualquer pessoa que tenha usado XMLHttpRequest, porém a nova API oferece um conjunto de recursos mais poderoso e flexível.
- Ao invocarmos fetch receberemos uma promise
  - **fetch(url, options)**
  - **url** é uma string com o endereço de nossa requisição
  - **options** é um objeto de configuração da requisição

# FETCH

```
// o método fetch nos retornará uma promise
fetch('https://viacep.com.br/ws/88032005/json')
  .then(resposta => {
    // podemos definir uma função then e testar se tivemos sucesso pelo atributo 'ok'
    if (resposta.ok) {
      // o método json da nossa resposta também é uma promise
      resposta.json();
      .then(conteudo => {
        // temos acesso ao nosso conteúdo JSON agora em forma de objeto JavaScript
        console.log(conteudo);
      });
    }
  });
```

Exemplo de uso de fetch





- Introduzidas no ECMAScript 2017, as palavras reservadas **async** e **await** nos permitem declarar e manipular funções assíncronas com mais facilidade.
- Para definir uma função como assíncrona, basta acrescentar a palavra **async** antes da definição.
  - **async function () { ... }**
  - **async () => { ... }**

- A palavra reservada **await** serve para “esperar” a conclusão de uma **Promise** e só pode ser utilizada dentro de uma função assíncrona.
  - **async** ( ) => {  
    **await** fetch('exemplo.json')  
    console.log('fetch concluido')  
}

- Como **await** é utilizado para aguardar a conclusão de uma **Promise**, o resultado dessa espera é o próprio retorno da **promise** resolvida.
- ```
async () => {  
    const response = await fetch('url')  
    console.log('Resposta', response)  
}
```

# ASYNC AWAIT

```
// promise com atraso
const delay = ms => new Promise(
  res => setTimeout(res, ms)
);

// uma arrow function assíncrona
const buscaCEP = async () => {
  // comando de espera
  await delay(1000);
  // executa após conclusão de delay
  console.log('Terminei de esperar');
}

// roda o exemplo
buscaCEP();
// imprime primeiro no console
console.log('Executei o console log');
```

Exemplo de criação de uma arrow function assíncrona

```
// promise com atraso
const delay = ms => new Promise(
  res => setTimeout(res, ms)
);

// uma função assíncrona
async function buscaCEP() {
  // comando de espera
  await delay(1000);
  // executa após conclusão de delay
  console.log('Terminei de esperar');
}

// roda o exemplo
buscaCEP();
// imprime primeiro no console
console.log('Executei o console log');
```

Exemplo de criação de uma função assíncrona

# TRY CATCH



# TRY CATCH

- Podemos nos perguntar: O que acontece quando a **promise** não é resolvida, mas sim rejeitada, quando ocorre uma exceção?
- Para estas situações, podemos contar com a estrutura de teste de exceções **try / catch**.
- Consiste em blocos de código encadeados:
  - **try**: bloco de código crítico a ser executado
  - **catch**: bloco executado quando ocorrer uma exceção em **try**
  - **finally**: sempre será executado ao final

# TRY CATCH

- O bloco **catch** espera um parâmetro, o valor deste parâmetro será qualquer exceção/falha ocorrida no bloco **try**:

- ```
async () => {  
  try {  
    const response = await fetch('http://site.com/endpoint')  
    console.log('Resposta', response)  
  }  
  catch (error) {  
    console.log(error)  
  }  
  finally {  
    console.log('Sempre executo ao final')  
  }  
}
```



# TRY CATCH

```
const buscaCEP = async () => {  
  try {  
    await delay(1000);  
    console.log('Terminei de esperar');  
  
  } catch (err) {  
    console.error(err);  
  }  
}  
  
buscaCEP();  
console.log('Executei o console log');
```

Exemplo de uso de try catch

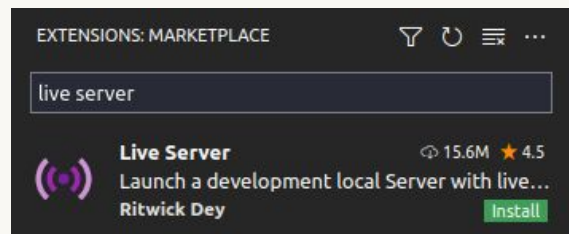
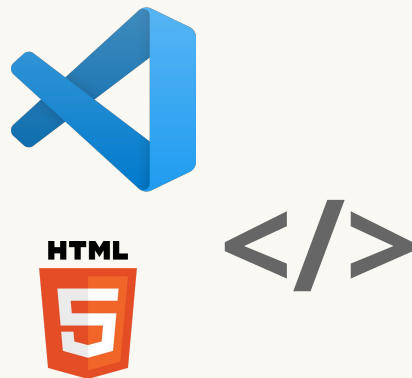
```
async function buscaCEPfetch() {  
  try {  
    const cep = campoCep.value;  
    const url = `https://viacep.com.br/ws/${cep}/json`;   
    const response = await fetch(url);  
    const data = await response.json();  
  
    console.log(data);  
    pTela.innerHTML = data.logradouro;  
  
  } catch (err) {  
    console.error(err);  
  }  
}
```

Exemplo de uso de try catch com fetch

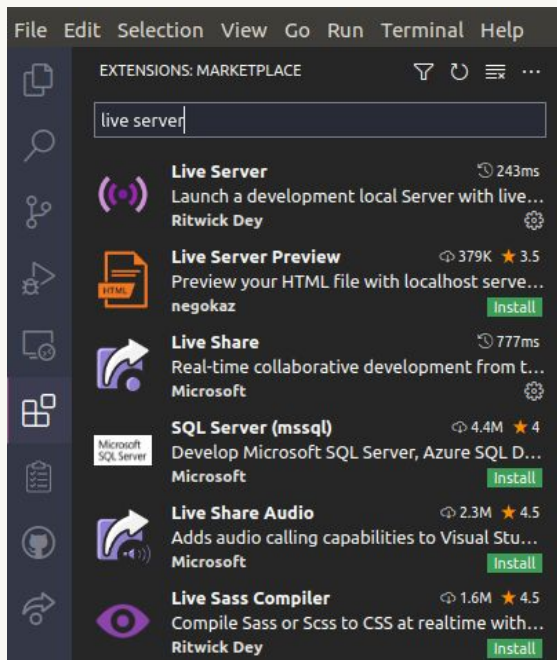
# PARA A MÃO NA MASSA

- Instalar **VS Code**  
(ou outro editor que se sentir mais confortável)  
<https://code.visualstudio.com>
- Sugestão: Instalar extensão **Live Server** no **VS Code**
- Criar um arquivo **index.html** no seu editor

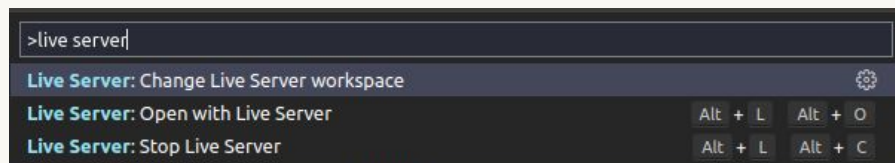
Code Sandbox | <https://codesandbox.io>  
PlayCode | <https://playcode.io/new>  
CodePen | <https://codepen.io/pen>  
JSFiddle | <https://jsfiddle.net>



# PARA A MÃO NA MASSA

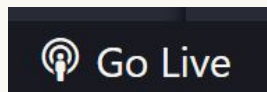


- **Ctrl + Shift + P**  
Live Server: Open with Live Server



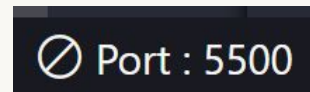
Start

- **Alt + L**
- **Alt + O**



Stop

- **Alt + L**
- **Alt + C**



# MATERIAL COMPLEMENTAR



Asynchronous JavaScript - Callbacks, Promises, Async/Await | <https://youtu.be/670f71LTWpM>

Learn Fetch API In 6 Minutes | <https://youtu.be/cuEtnrL9-H0>

06/10 Fetch - JavaScript Antes do Framework (React ou Vue.js) | <https://youtu.be/fhIDgAfuJZ8>

Como usar Async/Await? Promises no JavaScript? | <https://youtu.be/q28lfkBd9F4>

Async/await: o que é e como funciona | JavaScript | [https://youtu.be/Zl\\_jF7umgcs](https://youtu.be/Zl_jF7umgcs)

Asynchronous JavaScript - Callbacks, Promises, Async/Await | <https://youtu.be/670f71LTWpM>

Tips For Using Async/Await in JavaScript | [https://youtu.be/\\_9vgd9XKIDQ](https://youtu.be/_9vgd9XKIDQ)

JavaScript Async Await | [https://youtu.be/V\\_Kr9OSfDeU](https://youtu.be/V_Kr9OSfDeU)

Tratamento de erros / try catch finally | <https://youtu.be/AWroMPi9PXU>

try, catch, finally, throw - error handling in JavaScript | <https://youtu.be/cFTFtuEQ-10>

# MATERIAL COMPLEMENTAR



Fetch API | [https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch_API)

Fetch API e o JavaScript | <https://www.braziljs.org/p/fetch-api-e-o-javascript?s=r>

Funções assíncronas | [developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/async function](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/async_function)

JS assíncrono Async Await | [developer.mozilla.org/docs/Learn/JavaScript/Asynchronous/Async\\_await](https://developer.mozilla.org/docs/Learn/JavaScript/Asynchronous/Async_await)

JavaScript Async | [https://www.w3schools.com/js/js\\_async.asp](https://www.w3schools.com/js/js_async.asp)

Try Catch | <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/try...catch>

JavaScript Errors - Throw and Try to Catch | [https://www.w3schools.com/js/js\\_errors.asp](https://www.w3schools.com/js/js_errors.asp)



# DEVinHouse

Parcerias para desenvolver a sua carreira

**OBRIGADO!**



<LAB365>