

# targettrust

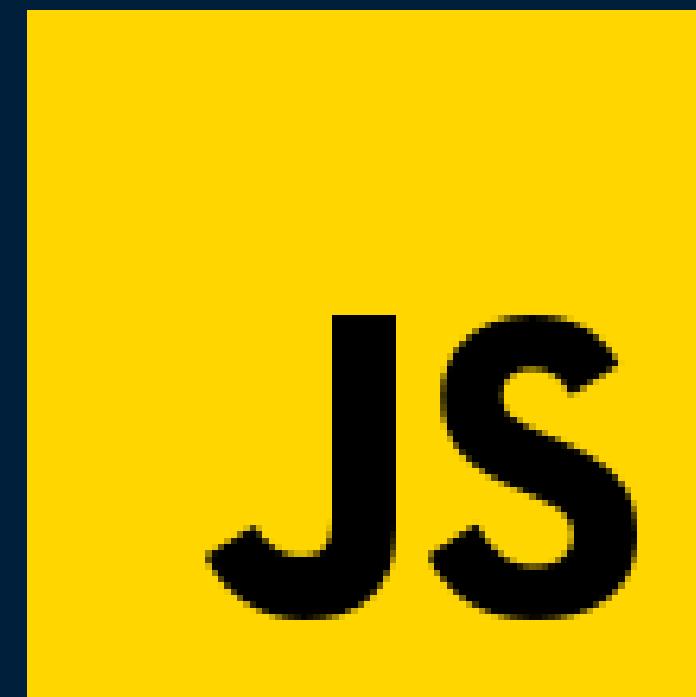
treinamento e tecnologia



# Apresentação

---

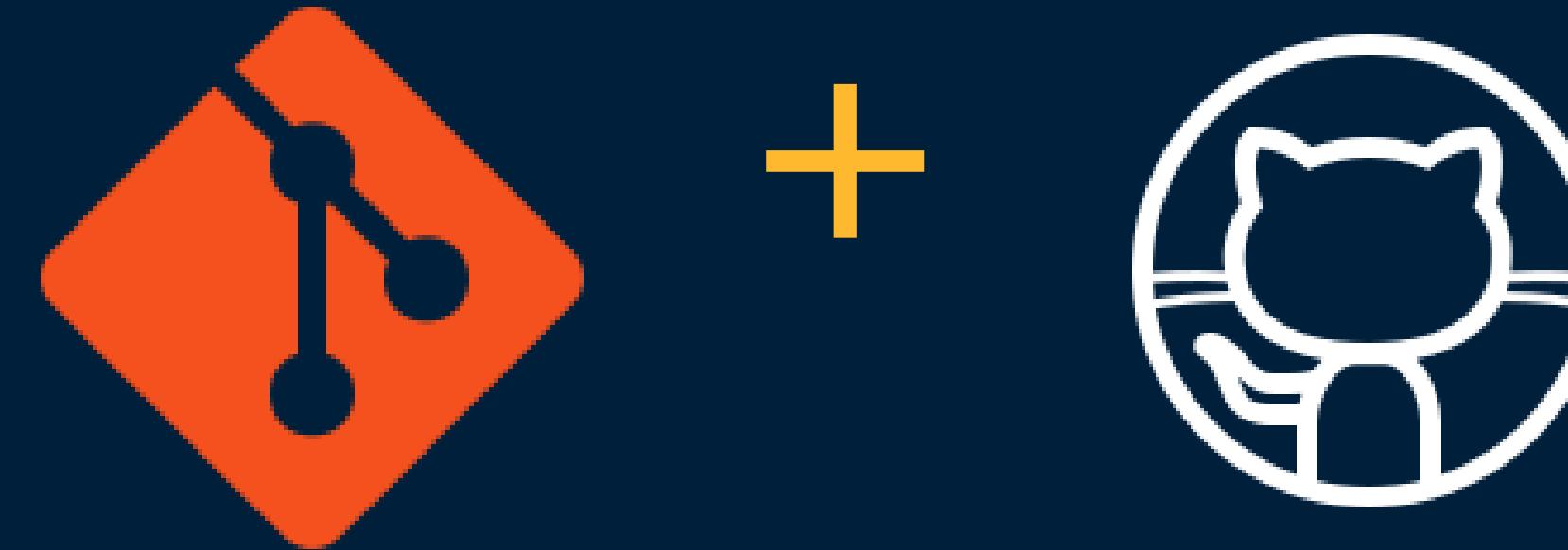
O que veremos:



# Revisão JavaScript

---

- Variáveis
- Condições
- Repetições
- Funções
- Eventos
- Dom



# Declarando Variaveis

Como declarar variaveis  
e seus tipos

```
var nome = "Abner Fonseca"; // String
var idade = 27; // Number
var possuiFaculdade = true; // Boolean
var time; // Undefined
var comida = null; // Null
var novoObjeto = {};// Object
```

Varificar o tipo de variavel

```
var nome = "Abner Fonseca";
console.log(">>", typeof nome);
```

# Concatenar Strings

Concatenando Strings

```
var nome = "Abner";
var sobrenome = "Fonseca";
var nomeCompleto = nome + " " + sobrenome;
```

Concatenando Strings com  
numero

```
var gols = 30;
var frase = "Eu chutei " + gols + " vezes";
```

Template String

```
var sobrenome = "Fonseca";
var nomeCompleto = `Abner ${sobrenome}`;
console.log(">>>, nomeCompleto);
```

# Boolean e Condicionais

Condicionais IF e else if

```
var possuiGraduacao = true;

if(possuiGraduacao) {
    console.log('Possui graduação');
} else {
    console.log('Não possui graduação');
}
// retorna Possui Graduação e não executa o else
```

# Boolean e Condicionais

Exemplo de Switch case

```
var corFavorita = "Azul";

switch (corFavorita) {
    case "Azul":
        console.log("Olhe para o céu.");
        break;
    case "Vermelho":
        console.log("Olhe para rosas.");
        break;
    case "Amarelo":
        console.log("Olhe para o sol.");
        break;
    default:
        console.log("Feche os olhos");
}
```

# Boolean e Condicionais

## Operadores de comparação

```
10 = 10; // true
10 === 10; // true
10 != 10; // false
10 !== 10; // false
'10' = 10; // true
'10' === 10; // false
```

# Funções

Função recebendo parametro  
retornando valor.

Dentro destas funções podemos  
ter condicionais e até mesmo  
chamar outras funções.

```
function areaQuadrado( lado ) {  
    return lado * lado;  
}  
  
areaQuadrado(4) // 16  
areaQuadrado(5) // 25  
areaQuadrado(2) // 4
```

Exemplo de Objeto e  
como acessar valores do  
objeto

```
var pessoa = {  
    nome: "Abner",  
    idade: 28,  
    profissao: "Fonseca",  
    possuiFaculdade: true  
};  
pessoa.nome; // 'Abner'  
pessoa.possuiFaculdade; // true
```

# Array e Loops

Como acessar valores no array,  
métodos e propriedade  
de uma array

```
var videoGames = ['Switch', 'PS4', 'XBox'];
videoGames[0] // Switch
videoGames[2] // Xbox
videoGames.pop(); // Remove o último item e retorna ele
videoGames.push('3DS'); // Adiciona ao final da array
videoGames.length; // 3
```

For - faz algo repetidamente  
até que uma condição seja  
atingida

```
for (var numero = 0; numero < 10; numero++) {
  console.log("">>>>", numero);
}
```

# Array e Loops

While - faz algo repetidamente até que uma condição seja atingida

```
var i = 0;
while (i < 10) {
  console.log(i);
  i++;
}
// Retorna de 0 a 9 no console
```

ForEach - faz algo executa uma função para cada item da Array. É uma forma mais simples de utilizarmos um loop com arrays

```
var videoGames = ["Switch", "PS4", "XBox", "3DS"];
videoGames.forEach(function (item) {
  console.log(item);
});
```

# Window e Document

window é o objeto global, por isso não precisamos chamar ele na frente de seus métodos.

Seleciona elementos por ID

Seleciona elementos por Class

Seleciona varios elementos com class diferentes

Seleciona elementos por TAG

```
//window é um objeto global  
window.alert('Isso é um alerta');  
alert('Isso é um alerta'); // Funciona  
document.querySelector('h1'); // Seleciona o primeiro h1  
document.body; // Retorna o body
```

```
const contatoSection = document.getElementById('contato');
```

```
const gridSection = document.getElementsByClassName('grid-section');
```

```
const contato = document.getElementsByName('grid-section contato');
```

```
const ul = document.getElementsByTagName('ul');
```

## Seletor geral único

```
const animais = document.querySelector(".animais");
const contato = document.querySelector("#contato");
const ultimoItem = document.querySelector(".animais-lista li:last-child");
const linkCSS = document.querySelector('a[href^="https://"]');
const primeiroUl = document.querySelector("ul");

// Busca dentro do Ul apenas
const navItem = primeiroUl.querySelector("li");
```

## Classlist

```
const menu = document.querySelector(".menu");

menu.className; // string
menu.classList; // lista de classes
menu.classList.add("ativo");
menu.classList.add("ativo", "mobile"); // duas classes
menu.classList.remove("ativo");
menu.classList.toggle("ativo"); // adiciona/remove a classe
menu.classList.contains("ativo"); // true ou false
menu.classList.replace("ativo", "inativo");
```

# Eventos

## ADDEVENTLISTENER e CALLBACK

```
const img = document.querySelector("img");
function callback() {
  console.log("Clicou");
}
img.addEventListener("click", callback); // 🚀
img.addEventListener("click", callback()); // undefined
img.addEventListener("click", function () {
  console.log("Clicou");
});
img.addEventListener("click", () => {
  console.log("Clicou");
});
```

## Elementos

createElement()

```
const animais = document.querySelector(".animais");
const novoH1 = document.createElement("h1");
novoH1.innerText = "Novo Título";
novoH1.classList.add("titulo");
animais.appendChild(novoH1);
```

# String

string.length

```
const comida = 'Arroz';
const frase = 'A melhor comida do mundo';
comida.length; // 5
frase.length; // 15
comida[0] // P
frase[0] // A
frase[frase.length - 1] // a
```

# String

---

## string.includes

```
const fruta = 'Banana';
const listaFrutas = 'Melancia, Banana, Laranja';
listaFrutas.includes(fruta); // true
fruta.includes(listaFrutas); // false
```

## String

## STR.SLICE(START, END)

```
const transacao1 = 'Depósito de cliente';
const transacao2 = 'Depósito de fornecedor';
const transacao3 = 'Taxa de camisas';
transacao1.slice(0, 3); // Dep
transacao2.slice(0, 3); // Dep
transacao3.slice(0, 3); // Tax
transacao1.slice(12); // cliente
transacao1.slice(-4); // ente
transacao1.slice(3, 6); // ósi
```

# String

## STR.REPLACE(REGEXP|SUBSTR, NEWSTR|FUNCTION)

```
let listaItens = 'Camisas Bonés Calças Bermudas Vestidos Saias';
listaItens = listaItens.replace(/\s+/g, ', ');
let preco = 'R$ 1200,43';
preco = preco.replace(',', '.'); // 'R$ 1200.43'
```

# String

STR.TRIM(), STR.TRIMSTART(), STR.TRIMEND()

```
const valor = ' R$ 23.00 '
valor.trim(); // 'R$ 23.00'
valor.trimStart(); // 'R$ 23.00 '
valor.trimEnd(); // ' R$ 23.00 '
```

## Var, Let, Const, Escopo

Escopo de variável

Escopo de função

```
function mostraCarro() {  
    var carro = "Fiat";  
    console.log(carro);  
}  
  
mostraCarro();  
console.log(carro);
```

## Var, Let, Const, Escopo

## Escopo de variavel VAR

Variavel vaza escopo

```
if (true) {  
    var carro = "Fiat";  
    console.log(carro);  
}  
  
console.log(carro); //fiat
```

## Var, Let, Const, Escopo

## Escopo de variavel LET

Variavel não vaza escopo ({} ) mas permite mudar valor.

```
if (true) {  
  let carro = "Fiat";  
  console.log(carro);  
}  
  
console.log(carro); //error
```

## Var, Let, Const, Escopo

## Escopo de variavel Const

Variavel não vaza escopo ({} ) e não permite mudar valor.

```
const carro = "Fiat";
console.log(carro);
carro = "Ford";
```

Inicie declarando a variável como Const e se necessário mude para let

# Arrow Function

Com a intenção de reduzir o tamanho do código em JavaScript foi criada uma nova sintaxe para as functions, chamadas de arrow functions.

```
function ola() {  
  console.log("Olá");  
}  
ola();  
  
const ola2 = () => {  
  console.log("Olá 2");  
}  
ola2()
```

# Arrow Function

Forma de passar argumentos ficou mais simplificada e o return pode ser simplificado somente para abertura/fechamento de escopo

```
const argumento = valor => {
    console.log(valor);
};

argumento("argumento");

const argumento2 = nome => (
    "olá " + valor
);

argumento2("Target")
```

# Input

Listar o objeto ao lado no html.  
usando forEach

Clicar no botao e mostrar alert de  
botao clicado.

Clicar no botao e listar Objeto com  
background vermelho e letras  
brancas

Clicar no botao e listar Objeto um do  
lado do outro.

```
const dados = {  
    alunos: [  
        { nome: "João", idade: 20 },  
        { nome: "Maria", idade: 25 },  
        { nome: "José", idade: 30 }  
    ]  
};
```

# Input

---

Pegando valor de input

```
const name = document.querySelector("#name").value;
```

Incluindo HTML

```
document.getElementById("listItens").innerHTML += `<div class="itemList">
<p> Teste numero ${i}</p>
</div>`;
`
```

# Exercício de Fixação

# Map

[].map(callback(itemAtual, index, array))

```
const carros = ["Ford", "Fiat", "Honda"];
const newCarros = carros.map((item) => {
  return "Carro " + item;
});
carros; // ['Ford', 'Fiat', 'Honda']
newCarros; // ['Carro Ford', 'Carro Fiat', 'Carro Honda'];
```

Map simples

Map com arrow  
function

```
//Map com Arrow Function
const numeros = [2, 4, 6, 8, 10, 12, 14];
const numerosX3 = numeros.map((n) => n * 3);
console.log(numerosX3);
```

# Map com objetos

[].map(callback(itemAtual, index, array))

Map com objetos

```
//map com objetos
const aulas = [
  {
    nome: "HTML 1",
    min: 15
  },
  {
    nome: "HTML 2",
    min: 10
  }
];

const tempoAulas = aulas.map((aula) => aula.min);
// [15, 10, 20, 25];

console.log(tempoAulas);

const puxarNomes = (aula) => aula.nome;
const nomesAulas = aulas.map(puxarNomes);
```

# Reduce

[].reduce(callback(acumulador, valorAtual, index, array), valorInicial)

Reduce com  
ternario

```
const numerosReduce = [10, 25, 60, 5, 35, 10];
const maiorValor = numerosReduce.reduce((anterior, atual) => {
    return anterior < atual ? atual : anterior;
});
console.log(maiorValor);
maiorValor; // 60
```

# Reduce passo a passo

[].reduce(callback(acumulador, valorAtual, index, array), valorInicial)

O primeiro parâmetro do callback é o valor do segundo argumento passado no reduce(callback, inicial) durante a primeira iteração. Nas iterações seguintes este valor passa a ser o retornado pela anterior.

```
const aulas = [10, 25, 30];
// 1
aulas.reduce((0, 10) => {
  return 0 + 10;
}, 0); // retorna 10

// 2
aulas.reduce((10, 25) => {
  return 10 + 25;
}, 0); // retorna 35

// 3
aulas.reduce((35, 30) => {
  return 35 + 30;
}, 0); // retorna 65
```

# Reduce passo a passo

`[].reduce(callback(acumulador, valorAtual, index, array), valorInicial)`

Se não definirmos o valor inicial do acumulador, ele irá **pular** a primeira iteração e começará a partir da segunda. Neste caso o valor do acumulador será o valor do item da primeira iteração.

```
const aulas = [10, 25, 30];  
  
// 1  
aulas.reduce((10, 25) => {  
  return 10 + 25;  
) // retorna 35  
  
// 2  
aulas.reduce((35, 30) => {  
  return 35 + 30;  
) // retorna 65
```

## Find e findIndex

## [].find() e [].findIndex()

].find(), retorna o valor atual da primeira iteração que retorna o valor ou item procurado. Já o ].findIndex(), ao invés de retornar o valor, retorna o index deste valor na array.

```
const frutas = ['Banana', 'Pêra', 'Uva', 'Maçã'];
const buscaUva = frutas.findIndex((fruta) => {
  return fruta === 'Uva';
}); // 2

const numerosfind = [6, 43, 22, 88, 101, 29];
const buscaMaior45 = numerosfind.find((x) => x > 45); // 88
```

## Filter

[].filter()

[].filter(), retorna uma array com a lista de valores que durante a sua iteração retornaram um valor verdadeiro.

```
const frutasFilter = ['Banana', undefined, null, '', 'Uva', 0, 'Maçã'];
const arrayLimpa = frutasFilter.filter((fruta) => {
  return fruta;
}); // ['Banana', 'Uva', 'Maçã']
const numerosFilter = [6, 43, 22, 88, 101, 29];
const buscaMaior45Filter = numeros.filter(x => x > 45); // [88, 101]
```

## Javascript Assíncrono

---

### Síncrono VS Assíncrono

#### ● Síncrono

Espera a tarefa acabar para continuar com a próxima.

#### ● Assíncrono

Move para a próxima tarefa antes da anterior terminar. O trabalho será executado no 'fundo' e quando terminado, será colocado na fila (Task Queue).

## Javascript Assíncrono

### Promisse

Promise é uma função construtora de promessas. Existem dois resultados possíveis de uma promessa, ela pode ser resolvida, com a execução do primeiro argumento, ou rejeitada se o segundo argumento for ativado

```
const promessa = new Promise(function (resolve, reject) {  
    let condicao = true;  
    if (condicao) {  
        resolve();  
    } else {  
        reject();  
    }  
});  
  
console.log(promessa);
```

### Promisse

- **Resolve()**

Podemos passar um argumento na função resolve(), este será enviado junto com a resolução da Promise.

- **Reject()**

Quando a condição de resolução da promise não é atingida, ativamos a função reject para rejeitar a mesma. Podemos indicar no console um erro, informando que a promise foi rejeitada.

- **Then()**

O poder das Promises está no método then() do seu protótipo. O Callback deste método só será ativado quando a promise for resolvida. O argumento do callback será o valor passado na função resolve.

## Javascript Assíncrono

## Promise + then() + catch()

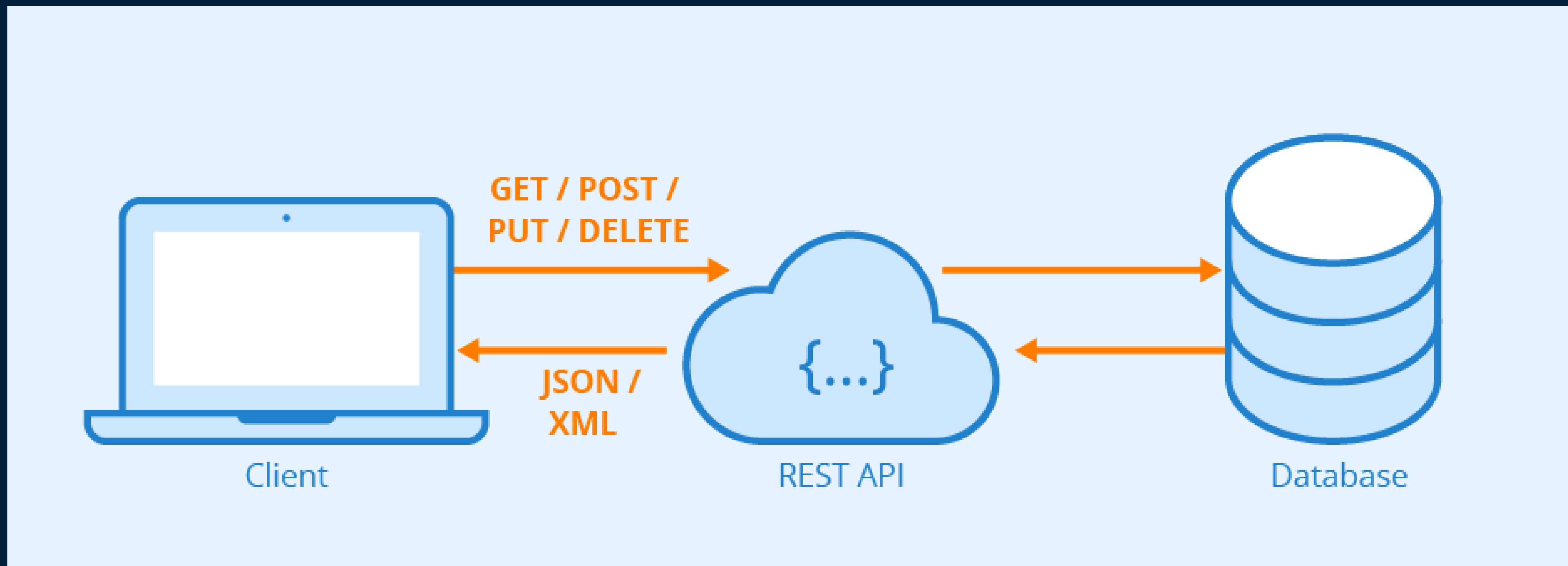
O método catch(), do protótipo de Promises, adiciona um callback a promise que será ativado caso a mesma seja rejeitada.

```
const promessaCatch = new Promise((resolve, reject)) => {
  let condicao = false;
  if (condicao) {
    resolve("Estou pronto!");
  } else {
    reject(Error("Um erro ocorreu."));
  }
};

promessa
  .then((resolucao) => {
    console.log(resolucao);
  })
  .catch((reject) => {
    console.log(reject);
  });

```

## O que é API?



### Methods

- Get      Use solicitações GET para recuperar apenas a representação/informações de recursos
- Post     POST para criar novos recursos subordinados
- Put      para atualizar um recurso existente (se o recurso não existir, a API poderá decidir criar um novo recurso ou não)
- Patch    são para fazer uma atualização parcial em um recurso.
- Delete   excluem os recursos

C Create  
R Read  
U Update  
D Delete

## Termos utilizados

- **endpoint** URL que será acessada para conectar ao serviço.
- **payload** corpo da requisição que será enviado ao serviço.
- **response** resposta da API para a sua consulta / requisição.

## Feth API

Permite fazermos requisições HTTP através do método `fetch()`. Este método retorna a resolução de uma Promise. Podemos então utilizar o `then` para interagirmos com a resposta, que é um objeto do tipo `Response`.

```
fetch("https://viacep.com.br/ws/01001000/json/")
  .then((response) => response.json())
  .then((cep) => {
    console.log(cep.bairro, cep.logradouro);
});
```

## Feth API - POST

```
fetch(urlBase, {
  method: "POST",
  headers: {
    Accept: "application/json",
    "Content-Type": "application/json"
  },
  body: JSON.stringify({ email: "abner.borda@gmail.com" })
})
  .then((response) => response.json())
  .then((data) => console.log(data));
```

## Apis para exemplo

---

### Lista de Apis

- mockApi
- viacep.com.br
- omdbapi.com

## PUT

```
fetch(`${urlBase}/${data.id}`, {  
  method: "PUT",  
  headers: {  
    Accept: "application/json",  
    "Content-Type": "application/json"  
  },  
  body: JSON.stringify(newTaskObjeto)  
})  
.then((response) => response.json())  
.then((data) => {  
  clearHtml();  
  listTack();  
});
```

## DELETE

---

```
fetch(`${urlBase}/${id}`, {
  method: "DELETE",
  headers: {
    Accept: "application/json",
    "Content-Type": "application/json"
  }
})
.then((response) => response.json())
.then((data) => {
  clearHtml();
  listTack();
});
});
```