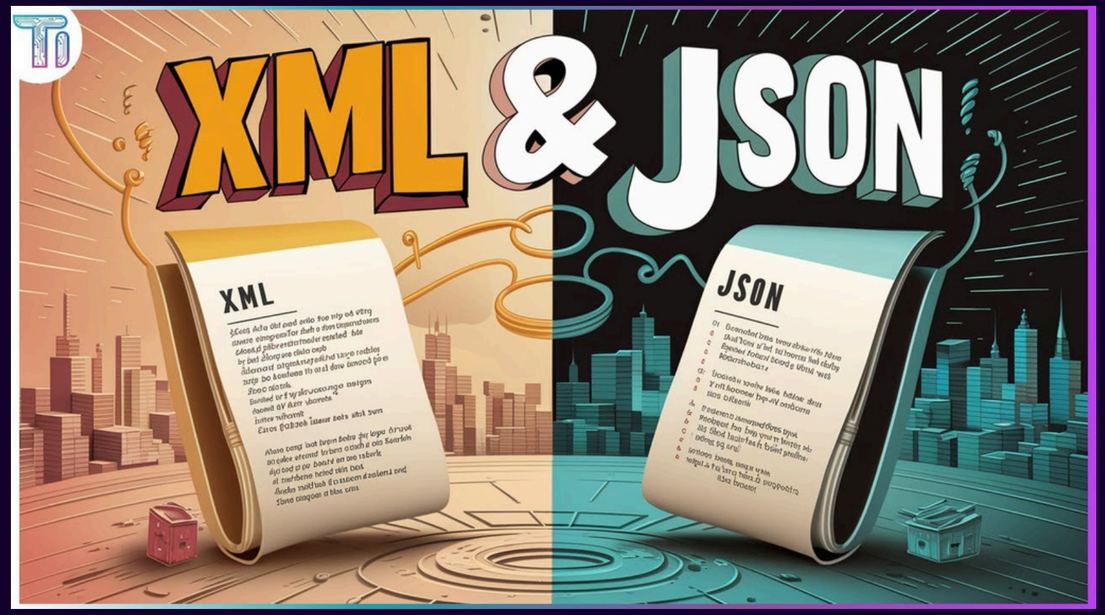

HELLO
WORLD

MANIPULAÇÃO DE JSON

A NATUREZA DOS OBJETOS EM JAVASCRIPT

O formato JSON (JavaScript Object Notation) é, como o nome sugere, uma forma de notação de objetos JavaScript, de modo que eles possam ser representados de uma forma comum a diversas linguagens.
Além disso, uma ideia que está fortemente enraizada neste formato é que ele seja facilmente trafegado entre aplicações em quaisquer protocolos, inclusive o HTTP.
Portanto, a principal diferença entre um objeto JavaScript padrão e um JSON é o fato do JSON ser na realidade: um texto.

```
{
   "cliente": {
       "id": 2020,
       "nome": "Maria Aparecida"
   },
   "pagamentos": [
       {
           "id": 123,
           "descricacao": "Compra do livro Cangaceiro JavaScript",
           "valor": 50.5
       },
       {
           "id": 124,
           "descricacao": "Mensalidade escolar",
           "valor": 1500
       }
   ]
}
```



O que é XML?
XML é um formato de dados que foi consolidado pelo W3C, sendo iniciados estudos em meados das décadas de 1990. O objetivo era criar um tipo de formato que poderia ser lido por software e que tivesse flexibilidade e simplicidade, visando, entre outras coisas:
– Possibilidade de criação de tags (você é quem cria as tags)
– Concentração na estrutura da informação e não em sua aparência

O que é JSON?
JSON, um acrônimo para “JavaScript Object Notation”, é um formato de padrão aberto que utiliza texto legível a humanos para transmitir objetos de dados consistindo de pares atributo-valor.

XML

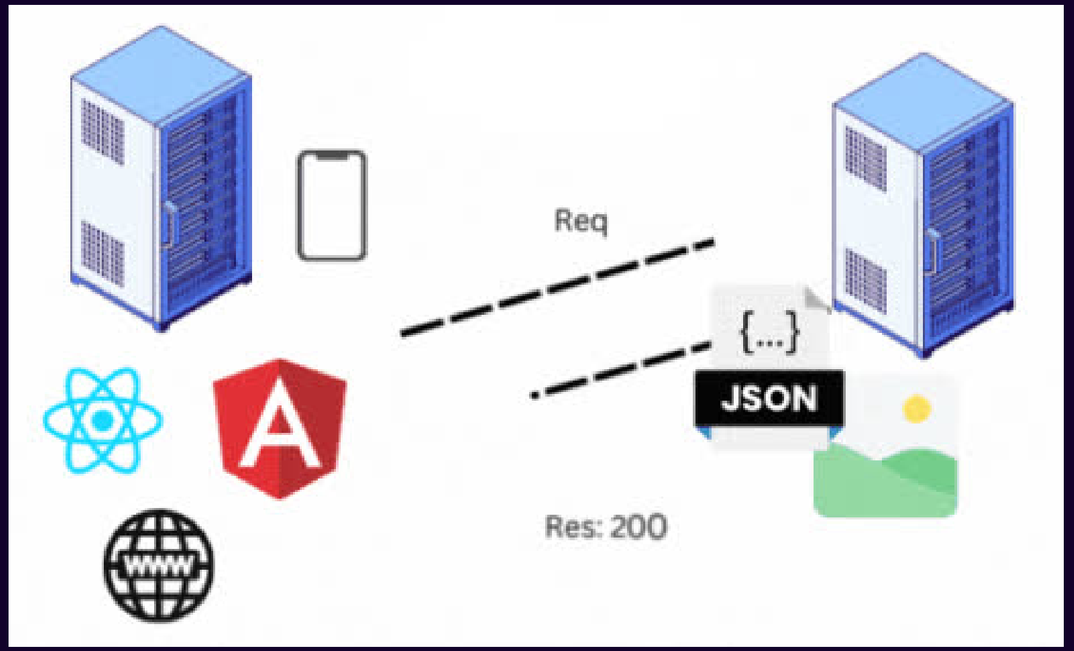
```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  “empinfo” :
    {
        “employees” :  [
        {
            “name” : “James Kirk”,
            “age” : 40,
        },
        {
            “name” : “Jean-Luc Picard”,
            “age” : 45,
        },
        {
            “name” : “Wesley Crusher”,
            “age” : 27,
        }
                        ]
    }
}
```

PARA QUE SERVE O JSON?

O JSON é um formato de dados leve e de fácil leitura utilizado para troca de informações entre sistemas computacionais. Ele é frequentemente usado para transmitir dados entre um servidor e um cliente em aplicações web e móveis, embora também seja utilizado em diversos outros contextos.
Ele é amplamente utilizado na web para representar dados estruturados de forma legível tanto para humanos quanto para máquinas. Em resumo, o JSON é uma forma popular de representar dados estruturados e transferi-los entre diferentes sistemas.



POR QUE O JSON É TÃO UTILIZADO?

A linguagem JSON é bastante usada, por oferecer simplicidade, legibilidade, portabilidade e suporte amplo, o que a torna uma escolha assertiva para a troca de informações na web e em outros ambientes.
Além disso, existem inúmeras outras razões pelas quais o JSON é amplamente utilizado. Confira algumas delas a seguir:

- **Simplicidade**: o formato JSON é relativamente simples e fácil de entender. Ele usa uma sintaxe leve e minimalista, tornando-o rápido de ser processado;
- **Legibilidade**: o JSON é projetado para ser legível tanto por humanos quanto por máquinas. Sua estrutura é organizada e fácil de analisar, facilitando a depuração de erros e o trabalho das pessoas desenvolvedoras;
- **Portabilidade**: ele é independente de plataforma e pode ser utilizado em diferentes linguagens de programação. Isso facilita o compartilhamento de dados entre sistemas heterogêneos, tornando o processo mais eficiente;
- **Suporte amplo**: a mai parte das linguagens de programação possui suporte nativo ou bibliotecas que facilitam a manipulação de dados em formato JSON. Isso torna mais simples o processo de codificação e decodificação de JSON em objetos ou estruturas de dados;
- **Integração com a web**: o JSON é muito utilizado na comunicação entre servidores e clientes em aplicações web, inclusive em APIs (Interface de Programação de Aplicativos), para transferir dados entre servidor e clientes de forma mais eficiente.


COMO USAR O JSON

Um JSON deve conter apenas informações que possam ser representadas em formato de texto. Listei algumas regras abaixo:

- Não pode ter funções;
- Não pode ter comentários;
- Todo texto sempre tem aspas duplas;
- As propriedades sempre tem aspas duplas.

Desta forma, imagine o envio do pagamento de uma nova fatura com o nome do cliente, um identificador numérico qualquer do cliente e uma lista de pagamentos a serem feitos na fatura em questão. Tais informações teriam, em JSON, o seguinte formato:

```
{
    "cliente": {
        "id": 2020,
        "nome": "Maria Aparecida"
    },
    "pagamentos": [
        {
            "id": 123,
            "descricacao": "Compra do livro Cangaceiro JavaScript",
            "valor": 50.5
        },
        {
            "id": 124,
            "descricacao": "Mensalidade escolar",
            "valor": 1500
        }
    ]
}
```

Embora se assemelhe com um objeto JavaScript literal, o JSON apresentado segue exatamente todas as regras que citei anteriormente. Além disso, é um formato muito mais simples e menos burocrático do que o mundialmente famoso XML que durante muito tempo, foi utilizado como padrão para o envio de informações entre aplicações.

Como você está usando JavaScript, Java e PHP nas aplicações em que estão envolvidos, trouxe um pequeno exemplo nestas linguagens:

```javascript
// JavaScript

// Criação do objeto fatura.
const fatura = {}

// Transforma o objeto literal em JSON.
const faturaJSON = JSON.stringify(fatura);

// Transforma o JSON em objeto literal.
const novamenteObjFatura = JSON.parse(faturaJSON);
```

```java
// Java usando a biblioteca Jackson

// Criação do objeto fatura.
Fatura fatura = {}

ObjectMapper mapper = new ObjectMapper();

// Objeto Java para JSON string.
String jsonString = mapper.writeValueAsString(fatura);

//JSON string para objeto Java.
Fatura novamenteFatura = mapper.readValue(jsonString, Fatura.class)
```

```php
// PHP

// Criação do objeto fatura.
$fatura = {}

// Transforma o objeto em JSON.
$faturaJSON = json_encode($meuObj);

// Transforma o JSON em objeto.
$novamenteObjFatura = json_decode($faturaJSON);
```

COMO MANIPULAR JSON NO JAVASCRIPT

Você pode trabalhar com JSON no JavaScript utilizando dois métodos principais: JSON.parse() e JSON.stringify().

- **JSON.parse()**: Converte uma string JSON em um objeto JavaScript.
- **JSON.stringify()**: Converte um objeto JavaScript em uma string JSON.

```
// String JSON
const jsonString = '{"nome": "João", "idade": 25, "hobbies": ["futebol", "música"]}';

// Convertendo JSON para Objeto JavaScript
const pessoa = JSON.parse(jsonString);

console.log(pessoa.nome); // João
console.log(pessoa.hobbies[0]); // futebol

// Modificando o Objeto JavaScript
pessoa.idade = 26;

// Convertendo de volta para JSON
const novaJsonString = JSON.stringify(pessoa);
console.log(novaJsonString);
```


FETCH API

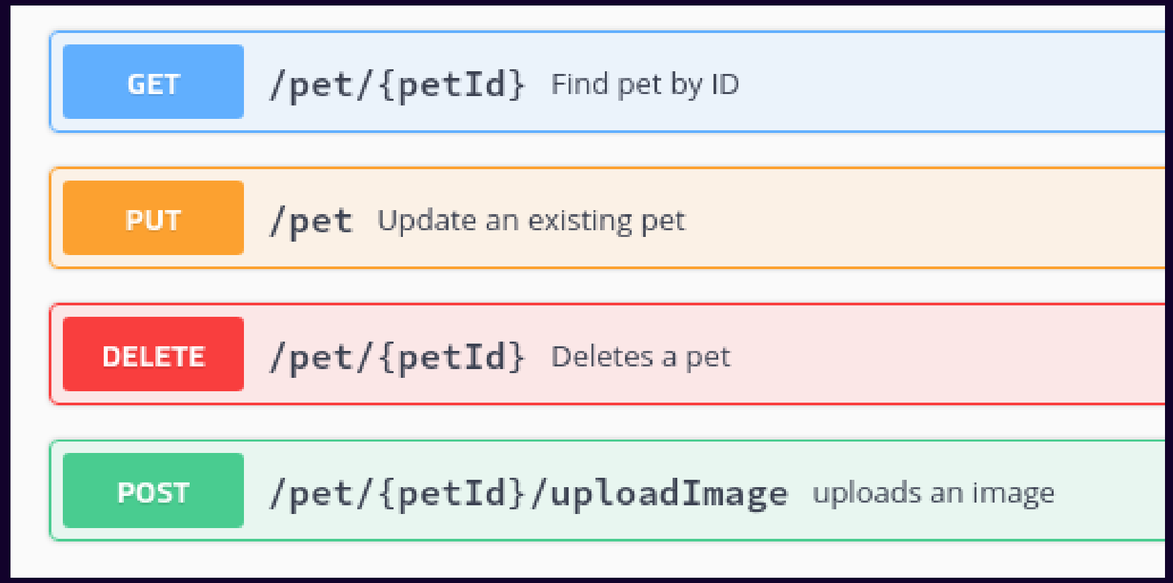
O QUE É A FETCH API?

A Fetch API é uma interface nativa do JavaScript usada para fazer requisições HTTP de forma assíncrona. Com ela, é possível consumir APIs e recursos de outros servidores sem que a página precise ser recarregada, permitindo uma experiência de usuário mais fluida e interativa.

Como Funciona?
O conceito principal por trás da Fetch API é que ela permite que você faça requisições assíncronas para servidores e trate as respostas utilizando Promises.
Uma requisição básica feita com fetch retorna uma Promise que será resolvida com a resposta da requisição.

```
1 fetch('https://api.exemplo.com/dados')
2   .then(response ⇒ response.json())  // Converter a resposta para JSON
3   .then(data ⇒ console.log(data))     // Exibir os dados no console
4   .catch(error ⇒ console.error('Erro:', error)); // Capturar erros
```

MÉTODOS DE REQUISIÇÃO



O fetch() suporta os principais métodos HTTP:

- GET: Para obter dados do servidor.
- POST: Para enviar dados para o servidor.
- PUT: Para atualizar dados no servidor.
- DELETE: Para excluir dados do servidor.

MANIPULANDO A RESPOSTA

Após realizar a requisição com fetch(), você obtém a resposta que pode ser manipulada de diversas formas. A resposta é um objeto Response que contém informações sobre o status da requisição e os dados retornados pelo servidor.

Verificando o Status da Requisição:
A fetch não rejeita a Promise automaticamente em caso de erro de status HTTP (exemplo, 404 ou 500). Por isso, é importante verificar o status da resposta:

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => {
    if (!response.ok) {
      throw new Error('Erro na requisição, status: ' + response.status);
    }
    return response.json();
  })
  .then(data => console.log(data))
```

TRATAMENTO DE ERROS

É fundamental tratar possíveis erros que possam ocorrer durante as requisições. Erros podem acontecer por vários motivos, como falha de rede ou problemas de CORS (Cross-Origin Resource Sharing).

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => {
    if (!response.ok) {
      throw new Error('Erro ao acessar os dados');
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => {
    console.error('Erro de rede ou de processamento:', error);
  });
```

CABEÇALHOS E CONFIGURAÇÕES AVANÇADAS

Os cabeçalhos HTTP informam detalhes sobre a requisição. Podemos configurá-los usando a opção headers no método fetch().

```
fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    title: 'Novo Post',
    body: 'Conteúdo do novo post',
    userId: 1
  })
})
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Erro:', error));
```


BORA
COOP!

MEUS CONTATOS:

27 99500-7495

https://beacons.ai/prismatech

producaoprismatech@gmail.com

Avenida Jerônimo Monteiro 145, Vitória


THANK
YOU