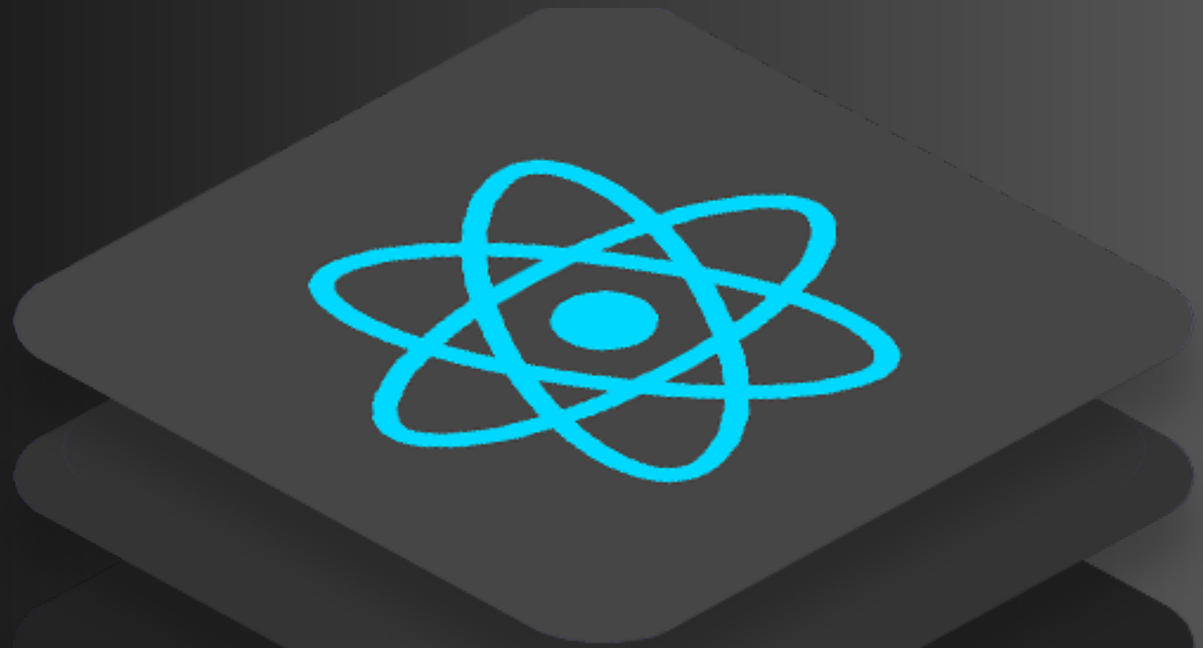

React

Frontend

API RestFul

API

Application Programming Interfaces - Permite a comunicação entre diferentes sistemas e aplicações

REST

Representational State Transfer - Baseado em:

- Utilização dos métodos HTTP (GET, POST, PUT, DELETE)
- Utilização de URL(Uniform Resource Locators) para identificar recursos específicos.
- Transferência de dados entre cliente e servidor usando formato padrão(JSON, XML ou outro)
- Manutenção do estado da aplicação no lado do cliente ao invés de manter no servidor.

RESTFUL

Mantém os critérios do REST de uma forma mais rigorosa:

- Interface uniforme: a API deve fornecer uma interface consistente e padronizada para acessar e manipular recursos.
- Clientes sem estado: cada solicitação enviada pelo cliente para o servidor deve conter todas as informações necessárias para entendê-la, sem depender de nenhum contexto armazenado no servidor.
- Operações baseadas em recursos: as ações realizadas pela API devem ser orientadas a recursos identificados por URLs únicas.

Método fetch()

Através do método fetch() fornecido pelo javascript o cliente pode acessar o servidor por meio de requisições e obter respostas

Esse método é assíncrono(Promise) e tem a estrutura:

```
let promessa = await fetch('url', [opcoes])
```

URL- Endereço da API para requisição de dados
opções:

- Method - GET, POST, PUT, DELET, PATCH. Por padrão se não informado é considerado GET
- Headers- Informações adicionais que o cliente pode passar sobre: recurso a ser obtido, informações de segurança entre outros.
- Payload - Dados que realmente interessam, nele está contido o body da requisição.

GET (READ)

```
let promessaDito = await fetch("https://pokeapi.co/api/v2/pokemon/ditto")
let dito = await promessaDito.json()
```

Promise encadeada

```
let ditoPokemon
fetch("https://pokeapi.co/api/v2/pokemon/ditto").then((promise)=>{
  promise.json().then((response)=>{
    ditoPokemon = response
  })
})
```

POST (CREATE)

```
let data = { name: "Fulano",avatar:"avatar1" }
let promessa = await fetch("url",
        { method: "POST",
        headers:{
          "Content-Type":"application/json"
        },
        body:JSON.stringify(data)
      }


)

let resposta = await promessa.json()
```

Formato de dados enviado nesse caso é o JSON

POST (CREATE)

Promise Encadeada

```
let data = { name: "Fulano",avatar:"avatar1" }
fetch("url",
        { method: "POST",
        headers:{
          "Content-Type":"application/json"
        },
        body:JSON.stringify(data)
        }
  ).then((resposta)=>
    resposta.json().then((objeto)=>
    console.log(objeto)
    )
  )
}
```

Formato de dados enviado nesse caso é o JSON

PUT (UPDATE)

Atualização completa do dado. Por exemplo:

Antigo:
usuario:
{nome:”Fulano”, idade:30, Nasc:10/12/1995
Novo:
usuario:
{nome:”Sicrano”, idade:31, Nasc:10/12/1994

Para modificação os dados precisam ser endereçadas a um identificador único.
Pode ser na URL:
“meuendereco.com/usuario/identificador-unico“

PATCH (UPDATE)

Atualização parcial do dado. Por exemplo:

No caso, pode ser modificado somente o campo nome ou outro campo. Um ou mais campos.

PUT (UPDATE)

```
let data = { name: "Fulano",avatar:"avatar1" }
let id = "identificador-único"
let resposta = await fetch(`url/usuario/${id}`,
        { method: "PUT",
        headers:{
            "Content-Type":"application/json"
        },
        body:JSON.stringify(data)
        }
    )
    let objeto = await  resposta.json()
}
```

PATCH (UPDATE)

```
let data = { avatar:"avatar1" }
let id = "identificador-único"
let resposta = await fetch(`url/usuario/${id}`,
        { method: "PATCH",
        headers:{
            "Content-Type":"application/json"
        },
        body:JSON.stringify(data)
        }
)
let objeto = await  resposta.json()
```

DELETE

```
let data = { avatar:"avatar1" }
let id = "identificador-único"
let resposta = await fetch(`url/usuario/${id}`,
        {
            method: "DELETE",
        }
)
let objeto = await   resposta.json()
```

Vamos à prática