
otterwise

Módulo Fundamentos Web

otterwise

JSON

otterwise

JSON

- JavaScript Object Notation.
- Formato de dados para interpretação e comunicação.
- Fácil entendimento tanto para pessoas quanto para máquinas.
- **Semelhante** ao Object do JS.
- Tipos permitidos:
  - string.
  - number.
  - boolean.
  - null.
  - object.
  - array.

Fonte: https://pt.wikipedia.org/wiki/JSON

Exemplo JSON

testes - data.json

```json
{
  "id": 1,
  "name": "Biblioteca do Juca",
  "books": [
    { "id": 1, "title": "Comunicação Não-Violenta" },
    { "id": 2, "title": "Obrigado pelo Feedback" },
    { "id": 3, "title": "Mindset" }
  ]
}
```

GERENCIADORES DE PACOTES

otterwise

Gerenciadores de pacotes

- Ferramentas que irão gerir os pacotes que temos no nosso projeto ou no nosso workspace (global).
- Utilizados para instalar, remover e atualizar nossos pacotes, em conjunto ou individualmente.
- Podem ser utilizados para executar scripts como build, start, etc...
- Exemplos: yarn, npm, etc...

package.json

- Arquivo responsável por informações pertinentes do projeto além de pacotes, dependências, scripts.
- Criando um arquivo **package.json** no seu projeto:
  - Utilizando o gerenciador **Yarn**: use o comando **yarn init** e siga os passos no prompt.
  - Utilizando o gerenciador **NPM**: use o comando **npm init** e siga os passos no prompt.

MÓDULOS

otterwise

Trabalhando com módulos

- Import e export em JS;
- Separar códigos e funções por contexto para reutilizá-los;
- Melhor legibilidade e manutenibilidade do projeto.

Exemplo de import/export nomeado ->

```
// hello.js

export const hello = () => console.log('Hello World')
```

```
// main.js

import { hello } from './hello.js'

hello() // "Hello World"
```

otterwise

Trabalhando com módulos

Exemplo de import/export padrão

```
// hello.js

const hello = () => console.log('Hello World')

export default hello
```

```
// main.js

import helloFunction from './hello.js'

helloFunction() // "Hello World"
```

EXERCÍCIOS

otterwise

Exercícios

1. Crie um arquivo chamado helpers.js e nele crie uma função que multiplique 2 números qualquer. Faça o export dessa função e em outro arquivo chamado index.js importe a função e a execute.

GERENCIAMENTO DE PACOTES

otterwise

Gerenciamento de pacotes

- Possibilidade de importar na nossa aplicação códigos de terceiro, disponibilizados publicamente, em forma de pacotes.
- Exemplo: **moment.js** e **date-fns** (pacotes de datas), **axios** (pacote para realizar requests HTTP), **react** (pacote para criação de interfaces de usuário), etc...
- Os pacotes vão facilitar o desenvolvimento, adicionando novas ferramentas e novas possibilidades para a aplicação.

Pacotes

- Quando instalamos o primeiro pacote é criado uma **pasta** chamada **node_modules** onde ficarão os códigos de terceiro que nosso projeto utiliza.
- Além disso um **arquivo lock** será criado para identificar a versão e a integridade dos pacotes instalados que estão dentro da **pasta node_modules**.
- Dentro do arquivo package.json ficará a versão do pacote que estamos usando no projeto. Essa informação pode ou não permitir que o pacote seja atualizado automaticamente.

Pontos de atenção

- Quando subimos nosso projeto para um repositório remoto **não** precisamos subir a pasta **node_modules**.
- Quando clonamos ou baixamos um projeto que possui um arquivo package.json utilizamos o comando **yarn** ou **npm install** para que os pacotes e dependências sejam instalados, criando assim a pasta **node_modules** e o arquivo **lock**.
- Tanto utilizando o Yarn ou NPM temos a pasta **node_modules**.
- O arquivo lock muda dependendo do gerenciador de pacotes que estamos utilizando, mas ele tem sempre o mesmo propósito:
- Com o Yarn o arquivo lock se chama **yarn.lock**.
- Com o NPM o arquivo lock se chama **package-lock.json**.
- Podemos instalar uma versão específica de um pacote.

EXERCÍCIOS

otterwise

Exercícios

1. Utilizando o **Yarn** ou **NPM**, crie o arquivo package.json e configure seu projeto.
2. Adicione ao seu projeto o pacote **date-fns**.
3. Crie um arquivo **formatters.js**:
    a. Nesse arquivo crie uma função chamada **formatDate** que recebe uma data como parâmetro e, utilizando a **date-fns**, retorna essa data no formato “DD/MM/YYYY”.
    b. Esse arquivo deve exportar a função formatDate.

Dica: para criar uma data com javascript utilize: new Date()

Contexto Web

Módulo WEB Developer

Comunicação com API’s

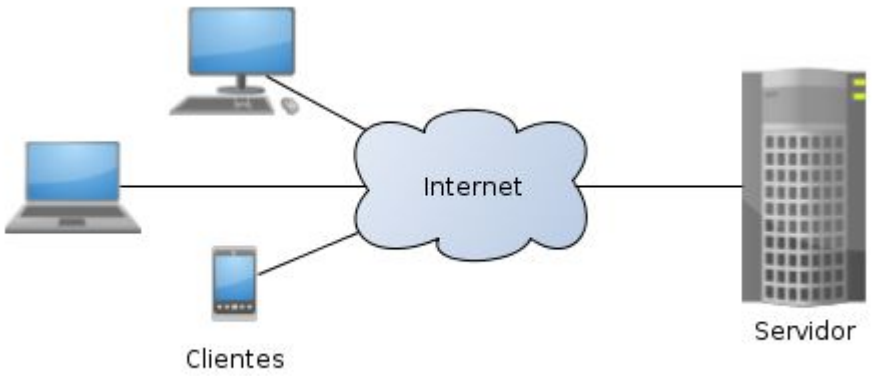
O que é uma API?

API é um conjunto de definições e protocolos usado no desenvolvimento e na integração de software de aplicações. API é um acrônimo em inglês que significa interface de programação de aplicações.

Fonte: https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces

Modelo Cliente Servidor

É o modelo onde vários clientes (computadores, celulares, etc) se comunicam com um servidor que fornece informações necessárias para esses clientes.



Fonte: https://pt.wikipedia.org/wiki/Modelo_cliente%E2%80%93servidor

Requisições

Requisições

Requisições servem para comunicar com um back-end onde podemos requisitar por um dado ou enviar dados para serem armazenados em um banco de dados.

Ao longo do curso vamos utilizar a biblioteca axios para fazer as nossas requisições. O axios é uma biblioteca que abstrai como uma requisição é enviado via método HTTP, que é um protocolo de comunicação de rede.

Promises

O axios é responsável por nos entregar uma Promise da requisição. Uma promise (de “promessa”) representa um valor que pode estar disponível agora, no futuro ou nunca.

Uma promise vai estar sempre em um destes estados:

- pending (pendente): Estado inicial, que não foi realizada nem rejeitada.
- fulfilled (realizada): sucesso na operação.
- rejected (rejeitado): falha na operação.

```
import axios from 'axios'

// pending
axios.get('https://url-da-api')
.then((response) => {
  // fullfilled
})
.catch((error) => {
  //rejected
})
```

```
import axios from 'axios'

const request = async () => {
  try {
    // pending
    await axios.get('https://url-da-api')
    // fullfilled
  } catch (error) {
    // rejected
  }
}

request()
```

Fonte: [Promise - JavaScript | MDN]

otterwise

Requisições

Para mantermos um padrão de comunicação entre front-end e back-end, precisamos seguir o padrão Representational State Transfer (REST). Neste padrão o servidor não mantém estado. Cada solicitação do cliente deve conter informações necessárias para o server entender a solicitação. O estado da sessão é mantido inteiramente no cliente.

Quando queremos requisitar dados de um back-end utilizamos os seguintes padrões a seguir:

Requisitando dados

Quando queremos requisitar um dado ao back-end, utilizamos o método HTTP **GET**, como é visto no exemplo abaixo utilizando axios.

À esquerda estamos requisitando um recurso, chamado **posts,** através do método HTTP **GET.**

```
import axios from 'axios'

const request = async () => {
  try {
    const response = await
axios.get('https://jsonplaceholder.typicode.com/posts')
    console.log(response)
  } catch (error) {
    console.log(error)
  }
}

request()
```

À direita estamos requisitando o mesmo recurso **posts**, mas agora requisitando apenas um elemento, pois estamos passando na URL um **identificador**. Assim, o back-end consegue identificar qual post deve ser retornado para a aplicação.

```
import axios from 'axios'

const request = async (data) => {
  try {
    const response = await
axios.get('https://jsonplaceholder.typicode.com/posts/1')
    console.log(response)
  } catch (error) {
    console.log(error)
  }
}

request()
```

Enviado dados para cadastro

Quando queremos enviar um dado ao back-end para ser cadastrado, utilizamos o método HTTP **POST**, como é visto no exemplo abaixo utilizando axios. Perceba que agora estamos chamando **axios.post**, assim o axios pode identificar que queremos fazer uma chamada HTTP do tipo **POST.**

```
import axios from 'axios'

const request = async (data) => {
  try {
    const response = await
axios.post('https://jsonplaceholder.typicode.com/posts', data)
    console.log(response)
  } catch (error) {
    console.log(error)
  }
}

const post = {
  title: 'Título do post',
  body: 'Corpo do post',
  userId: 1
}

// Agora precisamos enviar um objeto com os dados que gostaríamos de
cadastrar
request(post)
```

otterwise

Editando dados

Quando queremos requisitar um dado ao back-end, utilizamos o método HTTP **PUT** ou **PATCH**.

À esquerda estamos editando um recurso, chamado **posts,** através do método HTTP **PUT.** Neste método estamos dizendo para o back-end que todas as informações enviadas sobre o post devem substituir as informações armazenadas.

```
import axios from 'axios'

const request = async (data) => {
  try {
    const response = await
axios.put('https://jsonplaceholder.typicode.com/posts/1', data)
    console.log(response)
  } catch (error) {
    console.log(error)
  }
}

const post = {
  title: 'Título do post',
  body: 'Corpo do post',
  userId: 1,
  id: 1
}

request(post)
```

À direita estamos editando um recurso, chamado **posts,** através do método HTTP **PATCH.** Neste método estamos dizendo para o back-end que **apenas** as informações enviadas sobre o post devem substituir as informações armazenadas.

```
import axios from 'axios'

const request = async (data) => {
  try {
    const response = await
axios.patch('https://jsonplaceholder.typicode.com/posts/1', data)
    console.log(response)
  } catch (error) {
    console.log(error)
  }
}

const post = {
  title: 'Título do post',
}

request(post)
```

Excluindo dados

Quando queremos excluir um dado, utilizamos o método HTTP **DELETE**, como é visto no exemplo abaixo utilizando axios. Perceba que agora estamos chamando **axios.delete**, assim o axios pode identificar que queremos fazer uma chamada HTTP do tipo **DELETE.**

```
import axios from 'axios'

const request = async () => {
  try {
    const response = await
axios.delete('https://jsonplaceholder.typicode.com/posts/1')
    console.log(response)
  } catch (error) {
    console.log(error)
  }
}


request()
```

otterwise

EXERCÍCIOS

otterwise

Exercícios

1. Crie um projeto novo e instale a biblioteca axios.
    a. Consuma a lista de posts através do endpoint https://jsonplaceholder.typicode.com/posts.
    b. Consulte o post com ID de valor 2.
    c. Cadastre um novo post enviando um objeto com os atributos title e body para o endpoint https://jsonplaceholder.typicode.com/posts.
    d. Faça a edição de um post enviando novos atributos title e body para o post com id 1.
    e. Faça a exclusão post com ID de valor 3.

Exercício

Exercício

- Utilizando o postman e a url da api https://jsonplaceholder.typicode.com:
  - Consuma a lista de posts da api;
  - Cadastre um novo post;
  - Edite as informações title e body de um post;
  - Edite apenas o título de um post;
  - Exclua um post;