**Javascript Avançado e Moderno – Aula 07**

**Aula Gerenciamento de Pacotes – Introdução**

Um gerenciamento de pacotes irá lhe possibilitar os seguintes tópicos:

* Possibilidade de importar da nossa aplicação códigos de terceiro, disponibilizados publicamente, em forma de pacotes
  + Exemplos de pacotes:
    - Pacotes de datas
      * Moment.js
      * Date-fns
    - Pacotes para realizar requests HTTP
      * Axios
    - Pacote para criação de interfaces do usuário
      * React
* Os pacotes vão facilitar o desenvolvimento, adicionando novas ferramentas e novas possibilidades para a aplicação

Os pacotes assim têm como grande objetivo de adicionar funcionalidades as nossas aplicações.

**Aula Gerenciadores de pacotes parte 1**

O que são?

* 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

**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

Responsável por identificar as configurações do nosso projeto, os pacotes que nosso projeto usa, os scripts que ele tem e ele é o arquivo que chamamos de arquivo de configuração ou packet.json

**Aula Gerenciadores de pacotes parte 2**

**Pacotes:**

* Ao instalarmos um pacote, é criado uma página chamada node\_modules. Nele ficarão os códigos de terceiro que nosso projeto utiliza.
* Será criado também um arquivo lock para identificar a versão e a integridade dos pacotes instalados que estão dentro da pasta node\_modules
* O arquivo packet.json fica com a versão do pacote que estamos usando no projeto. Ela é capaz de permitir ou não que o pacote seja atualizado de forma automática

O node\_modules tem todos os módulos e pacotes que estamos utilizando.

O arquivo package-lock.json ele indica o que está instalado no node\_modules, ele é tipo um índice para o node\_modules. Serve para testar a integridade do pacote em caso de atualização

**Gerenciadores de Pacotes parte 3**

Quando clonamos ou baixamos um projeto que possui um arquivo package.json, é necessário que se utilize o comando npm install ou yarn install para que os pacotes e dependências sejam instalados, criando assim a pasta node\_modules e o arquivo lock

**Gerenciadores de Pacotes parte 4**

Podemos criar um script para criar um projeto. Dentro do script iremos utilizar comandos que fazem parte do SO que você estiver utilizando. Os comandos que tem do Yarn ou do Node

Exemplo: Vamos criar um script onde iremos substituir o node + nome do arquivo, para que tenhamos q rodar nosso arquivo no node, pelo script que nós criamos, veja:

"scripts": {

    "start": "node index.js"

  },

Para rodar agora iremos rodar o comando npm start

Veja:

Texto

Descrição gerada automaticamente

**Aula Módulos (export e import)**

Os módulos servem para isolarmos parte do código e reutilizarmos em outros arquivos, e iremos fazer isso não necessariamente utilizando uma função.

Aqui nós teremos uma melhor legibilidade e manutenibilidade do projeto

É muito melhor termos 5 arquivos com 100 linhas, do que um arquivo inteiro com 500 linhas, onde quando trabalhamos com os módulos cada módulo irá trabalhar apenas dentro de seu escopo, o que irá facilitar a legibilidade e manutenção desse código.

Para exportarmos uma função podemos exportar elas como padrão através do seguinte comando:

export default nome da função ou constante ou objeto, ou array...

function formatterCurrency(*value*) {

  return *value*.toFixed(2)

}

export default formatterCurrency

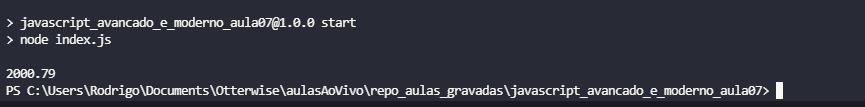
No arquivo que queremos receber esse export, ao utilizamos o export default, iremos utilizar o import, o nome da função ou elemento que queremos importar e definiremos a sua pasta.

Veja abaixo:

import formatterCurrency from './module.js'

const result = formatterCurrency(2000.787)

console.log(result)



Quando exportamos como default, tanto faz o nome que iremos importar, pois o que estamos buscando mesmo é a função que foi exportada em nosso outro arquivo, veja.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Isso acontece por estarmos trabalhando com os escopos do arquivo, onde o arquivo module é onde temos o escopo da função formatCurrency que está definida como uma função exportável.

No index.js nós teremos apenas em seu escopo uma instrução que será feita com a utilização do module.js

Um outro padrão é utilizarmos um export somente de uma função e não do seu arquivo como completo por conta do export default. Veja:

Texto

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Podemos também renomear da seguinte forma:

import { *formatCurrency* as oldFormat } from './module2.js'

import { formatQt } from './module2.js'

const result = oldFormat(6.68994)

const result2 = formatQt(6.68994)

console.log(`

Padrão atual de preço de combustíveis no Brasil R$ ${result2}`)

console.log(

  `

  Padrão de preço de combustíveis que será adotado no Brasil R$ ${result}

  `

)

Texto

Descrição gerada automaticamente

**Aula JSON**

É uma sigla para JavaScript Object Notation, ele é um formato de dados para interpretação e comunicação. Ele é muito semelhante ao objeto no JS. Mas ele é um tipo de arquivo no estilo XML que monta um estilo de padrão de dados, fácil de ler, seja para pessoas, como para máquinas.

Ele permite todos os tipos praticamente

* String
* Number
* Boolean
* Null
* Object
* Array

Texto

Descrição gerada automaticamente

import data from './data.json'

console.log(data)

{

  "name": "Rodrigo",

  "age": 36

}

Teremos um erro aqui, e para ele funcionar, iremos por um comando em nosso script, veja:

 "scripts": {

    "start": "node --experimental-json-modules index4.js"

  },

Texto

Descrição gerada automaticamente

Também podemos trabalhar com os objetos e arrays, veja:

import data from './data.json'

console.log(data)

console.log(data.name)

Texto

Descrição gerada automaticamente com confiança média

**Exercícios**

Crie um arquivo formatters.js

1. Nesse arquivo crie uma função chamada formatDate que receba uma data como parâmetro e, utilizando a date-fns, retorna essa data no formato “DD/MM/YYY.
2. Esse arquivo deve exportar a função formatDate

Crie um arquivo index.js

1. Nesse arquivo importe a função formatDate do módulo formatters.js
2. Imprima no console o resultado da chamada da função formatDate passando como argumento o dia de hoje (dica: new Date()).

**Promises**

Para registrar um callback, basicamente o que estamos dizendo é: "Quando função X acabar, execute Y e me dê o resultado". Então estamos delegando a resolução de uma computação para outro método.

Uma promise defie uma ação que vai ser executada no futuro, ou seja, ela pode ser resolvida (com sucesso) ou rejeitada (com erro).

Toda promise retorna um método then e outro catch, utilizamos o then para tratar quando queremos resolver a promise, e o catch quando queremos tratar os erros de uma promise rejeitada. Tanto then quanto catch retornam outra promise e é isso que permite que façamos o enacadeamento de then.then.then

Para criarmos uma Promise é muito simples, basta inicializar um new Promise que recebe uma função como parâmetro, esta função tem a assinatura (resolve, reject) => {}, então podemos realizar nossas tarefas assíncronas no corpo desta função, quando queremos retornar o resultado fazemos resolve e quando queremos retornar um erro fazemos reject

**Estados de uma Promise**

Uma Promise pode assumir quatro estados principais:

* Pending: O estado inicial da Promise, ela foi iniciada mas ainda não foi realizada nem rejeitada
* Fulfilled: Sucesso da operação, é o que chamamos de uma Promise realizada (ou, em inglês, resolved) — eu, pessoalmente, prefiro o termo resolvida.
* Rejected: Falha da operação, é o que chamamos de uma Promise rejeitada (em inglês, rejected)
* Settled: É o estado final da Promise, quando ela já sabe se foi resolved ou rejected

Uma Promise que está pendente (pending) pode vir a se tornar uma Promise resolvida com um valor, ou então rejeitada por um motivo (que é o erro).

Sempre que qualquer um dos dois casos acontecer, o método then da Promise será chamado e ele será o responsável por verificar se houve um erro ou um sucesso, chamando o método resolve em caso de sucesso, ou reject em caso de falha.

Diagrama

Descrição gerada automaticamente