**CREANDO PROJECTO EM NODE.**

- npm init

[ crea um package.json] y creas um archivo tipo index.js

PARA EJECUTAR EL ARCHIVO:

- node nombredelarchivo.js O node –watch nombredelarchivo.js

**ALGUNOS MODULOS QUE YA VIENEN EN NODE**

Para interactuar con el sistema de archivos: modulo file system [fs]

SIEMPRE HAY QUE ‘HABILITARLO’ CON require y guardarlo en una const.

const fs = require(‘fs’)

Y LUEGO SIMPLEMENTE USO CUALQUIERA DE LOS TANTOS METODOS

Ej fs.readFile() o el que sea.

PARA RECIBIR DATOS VIA TERMINAL

- npm install readline-sync

Y LO MISMO: LO HABILITO Y LUEGO LO USO

Ej const readlineSync = require(‘readline-sync’)

let nomeInserido = readlineSync.question(“Qual e seu nome¿”)

**VARIABLES DE AMBIENTE CON DOTENV**

- dotenv es una herramienta par acargar VARIABLES DE AMBIENTE y se usa normalmente par adatos sensibles como senhas de acceso, chaves y otras info sensibles.

PRIMERO HAY QUE INSTALAR LA BIBLIOTECA:

**npm i dotenv**

y luego creamos un archivo **.env** donde guardaremos esas variables de ambiente

luego necesitamos ‘activar’ ese modulo:

const dotenv = require("dotenv"); dotenv.config()

OU

require('dotenv').config();

y luego acessamos a ellas:

const minhaSenha = process.env.MINHA\_SENHA

const aulaHoje = process.env.AULA\_HOJE

**CREANDO UM SERVIDOR EXPRESS.**

PRIMERO INSTALARLO EM EL DIRECTORIO EM EL CUAL VAMOS A TRABAJAR:

npm install express

hay que ‘habilitarlo’ o ‘importarlo’:

const express = require (‘express’)

Y COMO express ES UN AFUNCION, LA GUARDAMOS EN OTRA VARIABLE app:

const app = express()

ACA PARA PODER YA TENER EL SERVIDOR ‘ANDANDO’ LOCAL EN POR EJ PORT 3000, USAMOS EL **METODO** listen que toma primero el servidor 3000 y luego un callback con una función que devuelve un mensaje en console.log:

app.listen(3000, () => console.log (‘O servidor está rodando’))

AHORA SI CORRES node archivo.js VA A ABRIR EL SERVIDOR EN PORT 3000.

La sintaxis entonces, ahora que ya tenemos el servidor creado, para definir METODOS y URL es al siguiente:

app.metodoHTTP(‘/URL’, función de callback) >>> eso es muy importante

DONDE:

app ES LA VARIABLE QUE CONTIENE A express() QUE ES QUIEN CORRE EL SERVER

metodoHTTP >> va a ser GET/POST/PUT/PATCH etc

‘/URL’ >> es el PATH o sea la ruta que vamos a seguir

función de callback >> va a depender de lo que querramos hacer

TRABAJANDO CON API

Las interacciones entre CLIENTE y SERVER son hechas por medio de REQUISICIONES **(req)** Y RESPUESTAS **(res)** USANDO EL PROTOCOLO **http** CON METODOS COMO **get, post** ETC.

Esto sería asi con su sintaxis: app.metodoHTTP(‘/URL path’, **callback**)

Yendo a las requisiciones y respostas quedaría algo asi:

app.get(‘/api’, **(req, res) => {**

**res.status(200).send(‘Olá Mundo “’)**

})

**ENTENDE QUE /api ES LA RUTA QUE YO QUIERA PONERLE, EL TEMA ES QUE SI YO PONGO app.get(‘/loquesea’…………**

**Luego en el browser o en Postman devo llamar a localhost:3000/loquesea**

app.post(‘/api’, **(req, res) => {**

**res.status(200).send(‘Requisição POST recibida!’)**

})

**ACORDATE QUE PARA TESTEAR LAS REQUISICIONES (get, post,etc) PODES USAR POSTMAN**

Ahora, para hacer **req**uisicioes podes agregarle una ruta(path) a la api para luego poner esos paramentros (**params)** en la **req.**

app.put(‘/api/**:aluno**’, (req, res) => {

const aluno = req.**params.aluno**

res.status(200).send(‘Seja bem-vindo ${aluno}’)

})

Entonces para checkear esa requisicion en POSTMAN colocamos:

http://localhost:3000/api/Ruan

Tambien se puede acceder con **query params:**

app.get(‘/api’, (req, res) => {

console.log(**req.query)**

res,status(200).send(‘Ola, ${**req.query.nome**}’)

})

Y LUEGO CONFERIMOS EN POSTMAN:

http://localhost:3000/api¿nome=Ruan

**PARA ACCESAR EL BODY.**

Cuando en requisiciones POST, PUT, PATCH con data sensible, se pasa la info en el **Body** en formato **JSON.**

**PARA ESO ES NECESARIO INSTALAR UN PACOTE body-parser**

**npm install body-parser**

Luego lo importamos:

const bodyParser = require(’body-parser’)

app.use(bodyParser.json())

Y luego usamos la propiedad de la requisicion: **req.body**

app.post(’/api’, (req, res) => {

const body = **req.body**

res.status(200).send(‘Recebemos seus dados ${**body.nome**}!’)

})

**HAY 3 MANERAS DE PASAR PARAMETROS EN CONSUTAS A API:**

Parametros de ruta (**req.params**) >>>> localhost:300**/api/:nomeDoParametro**

localhost:3000**/api/3**

Parametros de consulta/query **(req.query)** >>>> localhost/api**¿usuário=ruan**

Parametros de body (**req.body**) >>>> **se pasan em el body, fuera de la URL**

**PARA ACCESAR EL BODY:**

Instalar siempre el pacote **body-parser >>> npm install body-parser**

Importarlo **>>> const bodyParser = require(‘body-parser’)**

App.use(bodyParser.json())

Utilizar la propiedad de requisicion **req.body >>> app.post(‘/api’, (req, res) => {**

**const body = req.body**

**res.status(200).send(‘Recebemos seus dados ${body.nome}!)**

**})**

**!!! IMPORTANTE !!!**

**HOY SE PUEDE ACCEDER AL BODY SIN LA NECESIDAD DE body-parser**

**BASTA SUSTITUIR body-parser.json() POR express.json()**

**TRABAJANDO CON imports Y exports.**

Para EXPORTAR un modulo >>> **module.exports**

Ej: si quiero exportar simplemente un ELEMENTO ( en esta caso la const “local”) hago asi:

**CON module,exports**

const local = ‘Senai’

**module.exports** = local

Y luego lo importo en otro archivo:

const local = **require(‘./arquivo’)**

**CON exports**

**exports.local =** ‘Senai’

Y lo importo (como un objeto) asi:

**const { local } = require(./arquivo’)**

**AHORA BIEN, LA MANERAD MAS ACTUAL DE HACER LAS imports Y exports ES INCLUIR EN NUESTRO package.json DEBAJO DE “main”: “index.js”,**

**“type”: “module”,**

**ENTONCES AHORA LA SINTAXIS QUEDA ASI:**

const local = ‘Senai’

const novo = ‘Novo’

**export { local, novo }**

**import { local, novo } from ‘./arquivo.js’**

**OUTRA MANERA ES IMPORTAR/EXPORTAR CON default [se usa solo en valores principales de un modulo, como función RAIZ, y solo podes tener 1 default por arquivo] –no usa llaves-**

const local = ‘Senai’

const novo = ‘Novo’

**export default** novo

**import** novo **from ‘./arquivo.js’**

**MODEL – VIEW – CONTROLLER.**

Es uma manera de SEGMENTAR nuestro código em distintas “CAPAS” de SERVIDOR, CONTROLLERS Y ROUTES.

Para eso vamos a usar el concepto aprendido de **imports y exports.**

**Primero: siempre acordarse de agregar en el package.json debajo de “main”:”server.js”**

**“type”: “module”,**

Entonces en nuestra carpeta de proyecyo vamos a tener:

El package.json

server.js

Un folder **controllers [ aquí algún archivo.js]**

Un folder **routes [ aquí algún archivo.js]**

**Vamos por partes:**

En el archivo.js de la carpeta **controllers** VAMOS A DEFINIR LAS **funciones** QUE SERAN RESPONSABLES POR RECIBIR LA **req**  Y DAR LA **res** AL USUARIO. Y LUEGO LAS **EXPORTAREMOS:**

const compras = [

    { id: 1, descricao: 'PS5', preco: 4000 },

    { id: 2, descricao: 'coxinha', preco: 7 }

]

const getCompras = (req, res) => {

    // Lógica necessária

    res.status(200).send({ dados: compras })

}

const deleteCompras =  (req, res) => {

    // Lógica necessária

    res.status(200).send({ mensagem: "Elemento excluido com sucesso" })

}

const postCompras = (req, res) => {

    // Lógica necessária

    res.status(201).send({ mensagem: "Produto adicionado com sucesso"})

}

export { getCompras, deleteCompras, postCompras }

**EXPLICACION:** tenemos en compras, 2 objetos. Luego tenemos 3 const (una para get, otra para delete y otra para post) que simplemente cada una envía una **res** con un **.send** y en el caso de getCompras manda dados: compras y en las otras dos manda un mensaje.

Fijate que al final usas **export { getCompras, deleteCompras, postCompras}** para exportar esas funciones.

En el archivo.js de **routes:** se definen las rutas que estarán disponibles y las acciones que ellas irán a activar (controllers que serán acionados).

Para eso usamos el método **express.Router()**

Entonces:

import express from 'express'

import { getCompras, deleteCompras, postCompras } from '../controllers/compras.js'

const router = express.Router()

router.get('/compras', getCompras)

router.delete('/compras/:id', deleteCompras)

router.post('/compras', postCompras)

export default router

EXPLICACION: siempre primero importamos express. Importamos también las 3 funciones que teníamos exportadas en Controllers (getCompras, deleteCompras y postCompras) >> fíjate que van entre {}.

Creamos **const router = express.Router()** >>> nos permite trabajar ahora con routes.

Y luego la sintaxis es:

**router.metodo(‘/path’, callback o sea función importada)**

Ej **router.post(‘/compras’, postCompras)**

Finalmente exportamos **router >>> export default router** o **export { router }**

POR ÚLTIMO, **server.js** (que es nuestro archivo ‘main’) será el encargado de iniciar el servidor y utilizar las rutas creadas:

import express from 'express'

import comprasRouter from './routes/compras.js'

const app = express()

app.use(express.json())

app.use(comprasRouter)

app.listen(3000, () => console.log('Servidor iniciado'))

EXPLICACION: siempre importamos express, importamos routes **[aca una salvedad: lo podemos exportar con un nuevo nombre, en este caso** comprasRouter, eso no importa, lo que importa es que siga el **path** correcto**]**. Guardamos express en **app**, y lo usamos con el **método .use()** en este caso:

**app.use(express.json()) >>> para que use json**

**app.use(comprasRoutes) >>> para que use las rutas.**

Y no olvidarse que en nuestro archivoMain.js (server.js en este caso), tener corriendo el servidor en port 3000