Recomendado baixar o VScode para a aplicação no node.

Para usar o node usamos o cmd ou processador de comandos do computador para abrir ou inserir a pasta que será usada usamos o comando cd e depois colocamos o local onde está a pasta ou simplesmente abrir o cmd digitar um cd e arrastar a pasta para o cmd que ela ficará aberta.

Agora dentro do cmd faça um “Hello world!” simples com javascript dentro do VScode.

Depois digite node e o nome do arquivo javascript:

C:\Users\PC\Documents\FrontEnd Roadmap\Node Js> node Primeiro.js

•Sistema de módulos

Se uma aplicação for muito grande é recomendado dividir os arquivos em partes.

Exemplo crie uma calculadora com as seguintes funções somar, subtrair, dividir e multiplicar.

let somar = function(a, b){

return a + b

}

Depois crie um module esta função serve para exportar a função para a calculadora.

module. exports = somar;

Depois de criada cada uma das funções da calculadora crie uma variável com a seguinte função:

let somaf = require (“. /somar”)

A função require serve para incluir módulos e ela vai ser bem recorrente.

•HTTP servidor

Para criar um servidor o node já vem com um modulo básico. Para criar um servidor http faça:

let http = require(‘http’);

Depois faça:

http. createServer. listen(8080);

O node já te da função createServer pronta para o uso. Usamos um listen para informar a porta que será usada o recomendado é usar uma porta com números maiores pois se uma porta já estiver em uso o servidor do node não vai funcionar.

Se você ligar o servidor agora depois de fazer isso ele vai ficar rodando pois não tem nada dentro dele.

Então para testar se o servidor realmente está ligado precisamos colocar uma função dentro dele.

http. createServer. (function( req, res ){

res.end (“Olá chefe!”)

) listen (8080);

A função recebe dois parâmetros req de requisão/request e res de reposta/response. Usamos ali o res.end para mostrar uma mensagem e de fato provar que o servidor está funcionando.

•Introdução ao Express

O express é um framework do node js com módulos para fazer um servidor de uma forma mais tranquila. Para instalar o express vamos no site da <https://www.npmjs.com/> e procuramos por ele. O npm é o maior repositório de pacotes e módulos para qualquer tecnologia no momento.

Primeiro olhamos se temos o npm instalado:

npm -v

Depois usamos o comando npm install [nome do pacote] isso tudo dentro da pasta onde está sendo feito o projeto.

npm install express – save

Isso tudo irá salvar os módulos e o pacote dentro da pasta do projeto e como é a primeira vez ele irá criar um arquivo. package-lock. JSON depois iremos mexer nele, mas por enquanto não exclua.

•Início Express

Para usar o express depois de instalado na sua pasta primeiro devemos criar uma variável e chamar o modulo do express:

let express = require(‘express’)

No require agora você não precisa usar um ./ para colocar o express.

Depois criar uma variável para chamar a função express:

const [nome do app/site] = express ();

O ideal e colocar dentro de uma constante porque se você sobrescrever a variável você irá perder seu aplicativo e com uma constante isso não é possível.

Para criar um servidor diferente do básico do node você só precisa chamar a sua variável constante e colocar um listen na frente:

site.listen(8081);

Só com esta chamada seu servidor já está ligado, porém esta função tem que ficar no final do programa pois tudo o que for colocado depois não irá funcionar.

Provavelmente estará dando um erro no localhost da porta que você informa assim:” CANNOT GET /” porque você ainda não indicou uma rota a ser seguida. O Express é um framework de rotas, uma rota é o caminho da sua aplicação.

Para criar uma rota fazemos:

[nome da const] . get( “/”, function(req, res){

res. send(“Oi”);

})

Agora com está rota conseguimos uma resposta. Mas e se você quiser criar um outro caminho para outra parte exemplo a página sobre.

A função send serve para escrever uma mensagem e mostrar na página e só pode ser usada uma vez numa call-back.

Criamos a mesma função de rota mudando apenas o que tem dentro das aspas:

[nome da const]. get (“/sobre”, function (req, res) {

res. send (“Esta é a página sobre!”);

})

•Parâmetros Backend

Parâmetros servem mostrar ou procurar dados específicos em um site/servidor:

[nome da const]. get (“/ola”, function (req, res) {

res. send (“”);

})

Para inserir um parâmetro em uma rota basta adicionar /:[nome do parâmetro] ali após a primeira rota.

Depois disso adicionar a função req.params que irá mostrar para qual cada parâmetro está mostrando:

O req é responsável por obter dados da requisição inclusive parâmetros.

[nome da const]. get (“/ola/:nome”, function (req, res) {

res. send (“req.params”);

})

Para mostrar um parâmetro especifico de uma call-back usamos req.params.[nome do parâmetro].

[nome da const]. get (“/ola/:nome/:cargo”, function (req, res) {

res. send (“req.params.nome”);

})

•Nodemon

O nodemon e uma ferramenta que ajuda e faz que você não precise ficar abrindo e fechando servidor. Para instalar digite no cmd:

npm install nodemon -g

Isso fara com que instale na sua maquina o -g serve para mostrar que instalou globalmente ou seja qualquer projeto poderá ser usado o nodemon.

Para utilizar o nodemon iremos parar de digitar node [nome do arquivo] e iremos começar a digitar:

nodemon [nome do arquivo]

•Inserindo o HTML

Para exibir arquivos html no express ao invés de usar o res.send usamos o res.sendFile que serve para enviar arquivos para as rotas determinadas. Também usamos uma variável nova a \_\_dirname que serve pra localizar o seu diretório que será usado na rota.

app.get("/",function (req,res){

res.sendFile(\_\_dirname + "/html/index.html")

})

Porém esse não será o modo mais recomendado para o html mais pra frente pois nele não tem como inserir o CSS junto do arquivo html.

•Node js Sequelize

O Sequelize é uma ferramenta baseada no Node js para Postgres, MySQL, MariaDB, SQLite and Microsoft SQL Server.

Para instalar o sequelize use o comando do npm:

npm install --save sequelize

Depois de instalar o sequelize instale o npm para o banco de dados que será usado.

npm install --save pg pg-hstore # Postgres

npm install --save mysql2

npm install --save mariadb

npm install --save sqlite3

npm install --save tedious # Microsoft SQL Server

Para criar uma conexão com um banco de dados precisamos:

const Sequelize = require('sequelize')

const sequelize = new Sequelize('[nome do banco usado]','[nome do adiministrador]','[senha do administrador]',{

host: "localhost",

dialect: '[qual banco será usado]'

})

Para autenticar o banco e ter certeza que ele está funcionando usamos uma promise:

sequelize.authenticate().then(function(){

console.log('Conectado com sucesso!')

}).catch(function (erro) {

console.log('Falha ao se conectar:'+erro)

})

Para criar uma tabela dentro do sequelize fazemos:

const [nome da tabela] = sequelize.define('[nome da tabela]',{

/\*Criando os dados da tabela\*/

[nome do dado]:{

type:Sequelize.[tipo do dado]

},

[nome do dado]:{

type:Sequelize. [tipo do dado]

}

})

Para executar e criar a tabela logo após a criação usamos a seguinte função:

[nome da tabela].sync({force:true})

Logo após criar a apague essa função pois senão ela irá duplicar a tabela.

Para criar os dados da tabela fazemos:

/\*Criando dados para a tabela usuarios\*/

[nome da tabela].create({

[nome do dado]:"BOM-DIA!",

[nome do dado]:"Olá, como está?"

})

•Template-Engine

Template engines foram criadas para resolver a falta de reaproveitamento e facilidade na manutenção. As templates engines usadas em javascript são: Mustache;

•Handlebars;

•doT;

•EJS;

•PUG;

•Jade Language;

•Squirrelly, entre outros.

Iremos usar o handlebars para aprender o básico de template engines.Para instalar o handlebars usamos o npm.

npm install --save express-handlebars

Para usar o handlebars fazemos:

const handlebars = require('express-handlebars')

Para criara configuração do handlebars fazemos:

app.engine('handlebars',handlebars({defaultLayout:'main'}))

app.set('view engine','handlebars')

Depois crie uma pasta na raiz do seu projeto com o nome views(tem que ser assim) depois dentro da pasta crie uma pasta chamada layouts depois crie um arquivo dentro de layouts com o nome main.handlebars. Caso você não tenha criado uma pasta layouts e so setar o dafaltLayout:false.

Depois de seguir esses passos crie uma rota com o express para mostrar o seu arquivo com o handlebars:

app.get(‘/’, function(req, res){

res.render(‘ [nome do arquivo.handlebars] ’)

})

No caso do seu arquivo ser um formulário você tem que definir dois atributos o method e o action .Dentro do html existem dos métodos de envio o método GET e o método POST. No método GET ele envia os dados pela url. E no método POST ele não passa os dados pela url fazendo dele mais confiável.

Para adicionar a requisição do formulário devemos fazer assim:

app.post('/adicionar',function (req,res) {

res.send(‘POST recebido com sucesso!’)

})

Porém precisamos de uma biblioteca nova para conseguir mostrar os arquivos o nome dela é body-parser que server pra receber qualquer tipo de dados de um formulário.

•Body-Parser

O body-parser é uma biblioteca que serve para a amostra de qualquer tipo de dados no html.

Para instalar o body-parser com o npm fazemos:

npm install –save body-parser

Depois de instalado criar uma variável para chamar o modulo:

const bodyParser = require(‘body-parser’)

E depois configurar ele:

app.use (bodyParser.urlencoded({extendend: false}))

app.use (bodyParser.json())

Agora para mostar o conteúdo que você cadastrou basta usar a seguinte estrutura:

Lembrando que num formulário para usar o body-parser seus inputs/textareas precisam de nomes.

app.post('/adicionar',function (req,res) {

res.send(‘1°: ’ + req.body.titulo + ‘ 2°: ’+req.body.conteudo)

})

•Estruturando um Banco de dados

Primeiro crie uma pasta models que vai armazenar todos os modelos e dentro da pasta crie um arquivo pro banco de dados:

banco.js

Dentro do arquivo banco.js recorte os arquivos do sequelize e coloque dentro dele:

const Sequelize = require('sequelize')

/\*Conectando com o banco de dados\*/

const sequelize = new Sequelize('test','root','12345',{

host: "localhost",

dialect: 'mysql'

})

E aqui criamos um arquivo para exportar o sequelize:

module.exports = {

Sequelize: Sequelize,

sequelize: sequelize

}

Supondo que estamos separando por módulos iremos criar outro arquivo dentro da pasta módulos o arquivo Post.js. Dentro desse arquivo crie um a variável para chamar a conexão do banco de dados.

const banco = require(‘./banco’)

Agora iremos definir o banco:

const Post = banco.sequelize.define('test',{

titulo:{

type:banco.Sequelize.STRING

},

conteudo:{

type:banco.Sequelize.TEXT

}

})

Depois de usar essa função transforme essa linha em um comentário ou apague pois se não toda vez que você executar o programa seu banco de dados irá duplicar.

//Post.sync({force:true})

Aqui criamos o modelo para exportar:

module.exports = Post

•Pegando dados do formulário e salvando no banco de dados

Primeiro criar uma variável que vai chamar o modulo de postagem, com isso você não precisará mais acessar o Post.js.

const Post = require(‘./models/Post’)

Dentro da rota de adicionar iremos mudar algumas coisas;

app.post('/adicionar',function (req,res) {

Post.create({

titulo:req.body.titulo,

conteudo:req.body.conteudo

}).then(function () {

res.redirect('/')

}).catch(function (erro) {

res.send('Ocorreu um erro: '+erro)

})

})

Para retornar todos os posts que cadastramos do banco de dados precisamos criar uma função dentro da rota que está redirecionando.

app.get('/',function (req,res) {

Post.findAll({order:[['id','DESC']]}).then(function (posts) {

res.render('home',{posts: posts})

})

})

Dentro do arquivo home.handlebars como cadastramos muitos arquivos podemos criar uma função handlebars. Para ordernar os arquivos dentro do post.findAll() você precisa abrir um objeto order e depois dois colchetes [[‘id’,’DESC|ASC’]] isso irá exibit os arquivos pela ordem.

{{#each posts}}

<small>{{dataValues.createdAt}}</small>

<h1>{{dataValues.titulo}}</h1>

<p>{{dataValues.conteudo}}</p>

<hr>

{{/each}}

Para deletar as postagens do sequelize precisamos criar um botão dentro do html para excluir mas pra isso precisamos referenciar com uma ancora para definir o que vamos excluir.

{{#each posts}}

<small>{{dataValues.createdAt}}</small>

<h1>{{dataValues.titulo}}</h1>

<p>{{dataValues.conteudo}}</p>

<a href="/deletar/{{dataValues.id}}"><button>Deletar</button></a>

<hr>

{{/each}}

Depois criar uma rota para deletar os arquivos que você cadastrou no banco de dados.E informar o que você quer deletar exemplo o id,nome,titulo...

Depois criar uma função destroy que é ela que vai excluir seus conteúdos do banco de dados.

app.get('/deletar/:id',function (req, res) {

Post.destroy({where:{'id':req.params.id}}).then(function(){

res.send('Postagem deletada com sucesso!')

}).catch(function (erro){

res.send('Esta postagem não existe!')

})

})

•Introdução ao MongoDb

O mongoDb é um banco de dados como o mysql porém com uma diferença é que não existem tabelas, os arquivos são salvos em documentos como esse:

{

name: “Gabriel”

status: “C”

age: 22

groups: [“politcs”,”news”]

}

Um conjunto de arquivos como esse apresentado acima é chamado de collection.

•Instalando o MongoDb

Para instalar o mongoDb entre no site <https://www.mongodb.com/> baixe e instale, depois entre no dico local C:,arquivos de programa e pesquise por mongoDb entre na pasta server e dentro desta pasta terá a versão do mongo e depois entre na pasta bin. Copie o endereço do diretório e registre no path do Windows.

Para fazer isso entre no meu iniciar clique com o botão direito em computador depois propriedades, depois configurações avançadas do sistema, variáveis de ambiente em seguida path, editar:

Repare que seu path se for windows10 estará um abaixo do outro e ser for Windows 7 estará em linha.

Se for win7 coloque um ‘;’ no final da linha e cole o arquivo clique em ok e pronto a primeira parte da instalação está feita.

Agora dentro do disco local c: crie uma pasta com o nome de data, essa pasta que irá ficar armazenada todos os documentos. Depois dentro da pasta data crie uma outra pasta chamada db.

Para testar o servidor agora entre no cmd e digite mongod para testar e pronto o servidor já esta funcionando e pra manter isso sempre use 2 cmds um para o servidor e outro para o uso de aplicações.

•Iniciando no mongoose

O mongoose é um modulo muito mais simples que o sequelize e muito mais recursos.

Primeiro instale o módulo do mongoose na pasta onde está seu projeto:

npm install –save mongoose

Depois ligue o servidor com o comando no cmd:

mongod

Depois dentro da pasta onde instalamos o modulo do moongose crie um arquivo .js .

Agora crie uma variável para usar o mongoose:

const mongoose = require('mongoose')

Depois precisamos configurar e criar a conexão com o mongoDb, para isso precisamos de um simples comando:

mongoose.connect(‘mongodb://localhost/[nomedobanco]’)

Como nós sabemos que precisamos de uma mensagem para saber se está conectado ou não criamos uma promise:

mongoose.connect(‘mongodb://localhost/[nomedobanco]’).then(()=>{

console.log('Conectado ao MongoDb...')

}).catch((error)=>{

console.log('Houve um erro ao se conectar ao MongoDb.'+error)

})

Para pequenos erros futuros e boas praticas é bom setar alguns objetos na conexão e uma variável global.

//Configuração mongoose

mongoose.Promise = global.Promise

mongoose.set('useNewUrlParser', true);

mongoose.set('useFindAndModify', false);

mongoose.set('useCreateIndex', true);

mongoose.set('useUnifiedTopology', true);

mongoose.connect(‘mongodb://localhost/[nomedobanco]).then(()=>{

console.log('Conectado ao MongoDb...')

}).catch((error)=>{

console.log('Houve um erro ao se conectar ao MongoDb.'+error)

})

•Criando um documento e povoando uma coleção

Depois de criar uma conexão temos criar um modelo para os documentos do mongoDb para isso criamos uma variável. Normalmente usamos variáveis singulares e com Schema no final para boas práticas.

const UsuarioSchema = mongoose.Schema({

})

Agora criamos os dados dentro dos documentos diferentes do sequelize que eram vários tipos de dados o mongoDb usa os tipos de dados com javascript. Dentro do documento colocamos o tipo do dado e se ele é obrigatório ou não usando a propriedade require se for true é obrigatória se for false não.

const UsuarioSchema = mongoose.Schema({

nome: {

type:String,

require:true

},

sobrenome: {

type:String,

require:false

},

email: {

type:String,

require:true

},

idade: {

type:Number,

require:true

},

pais:{

type:String,

require:false

}

})

Assim criamos o primeiro modelo no mongoDb, agora para definir uma collection precisamos usar o modulo para exportar.

mongoose.model('Usuarios',UsuarioSchema)

Para povoar o documento criado precisamos importar o modelo criado acima assim;

const user = mongoose.model('Usuarios')

Agora para criar novos documentos precisamos criar um novo usuário e finalizar com um método para salvar.

new user({

nome:"Gabriel",

sobrenome:"Silva",

email:"gabrielsilva291996@gmail.com",

idade: 23,

pais:"Brasil"

}).save()

Mas para ter certeza se realmente inserimos uma pessoa na collection fazemos uma promise:

new user({

nome:"Gabriel",

sobrenome:"Silva",

email:"gabrielsilva291996@gmail.com",

idade: 23,

pais:"Brasil"

}).save().then(()=>{

console.log('Usuario criado com sucesso.')

}).catch((erro)=>{

console.log('Houve um erro ao registrar o usuário:'+erro)

})

Para olhar o documento dentro do mongo utilizamos o comando:

mongo

Depois o comando:

show databases;

Para ver quais tabelas foram ciradas, e para usar o banco usamos o comando:

use [nomedobanco];

Para mostar as collections usamos o comando:

show collections;

E para olhar o conteúdo do documento usamos:

db.[nomedacollection].find();