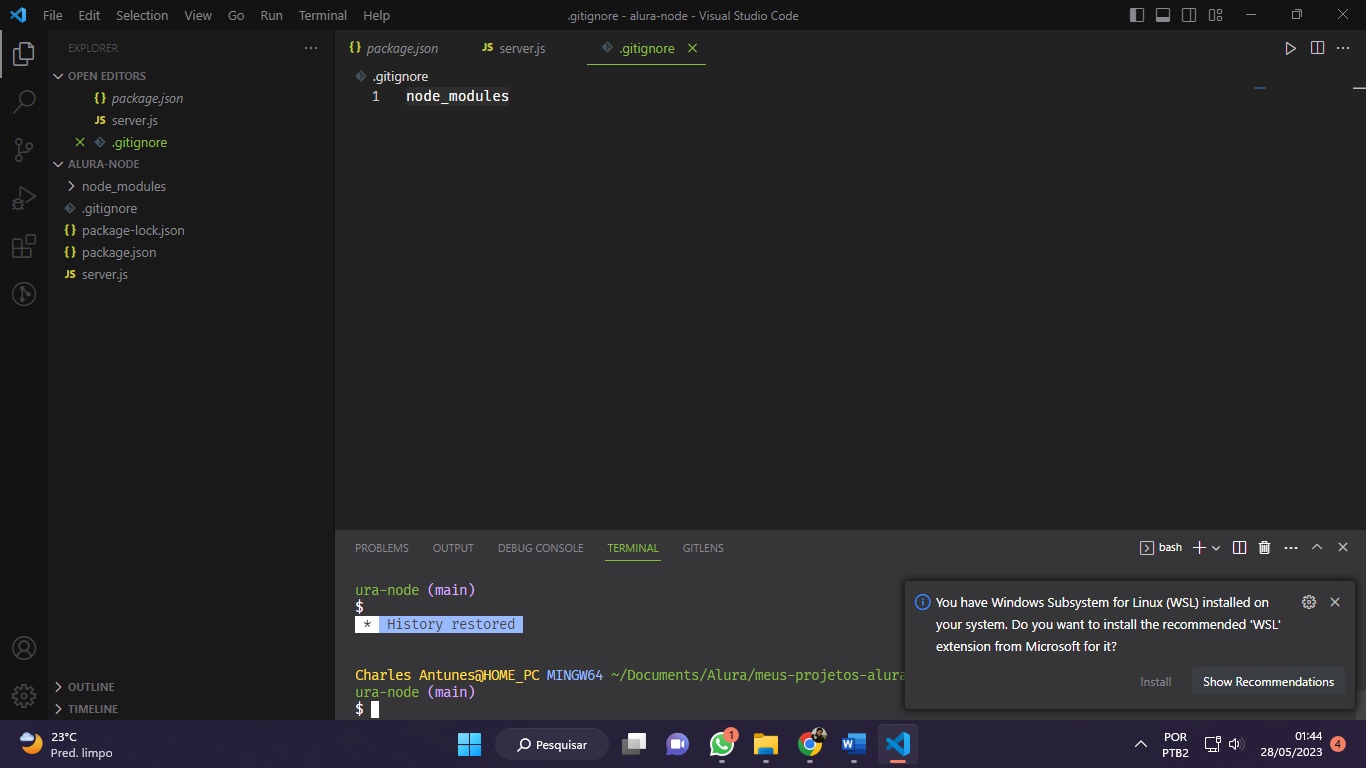
================================ Projeto API Rest========================================

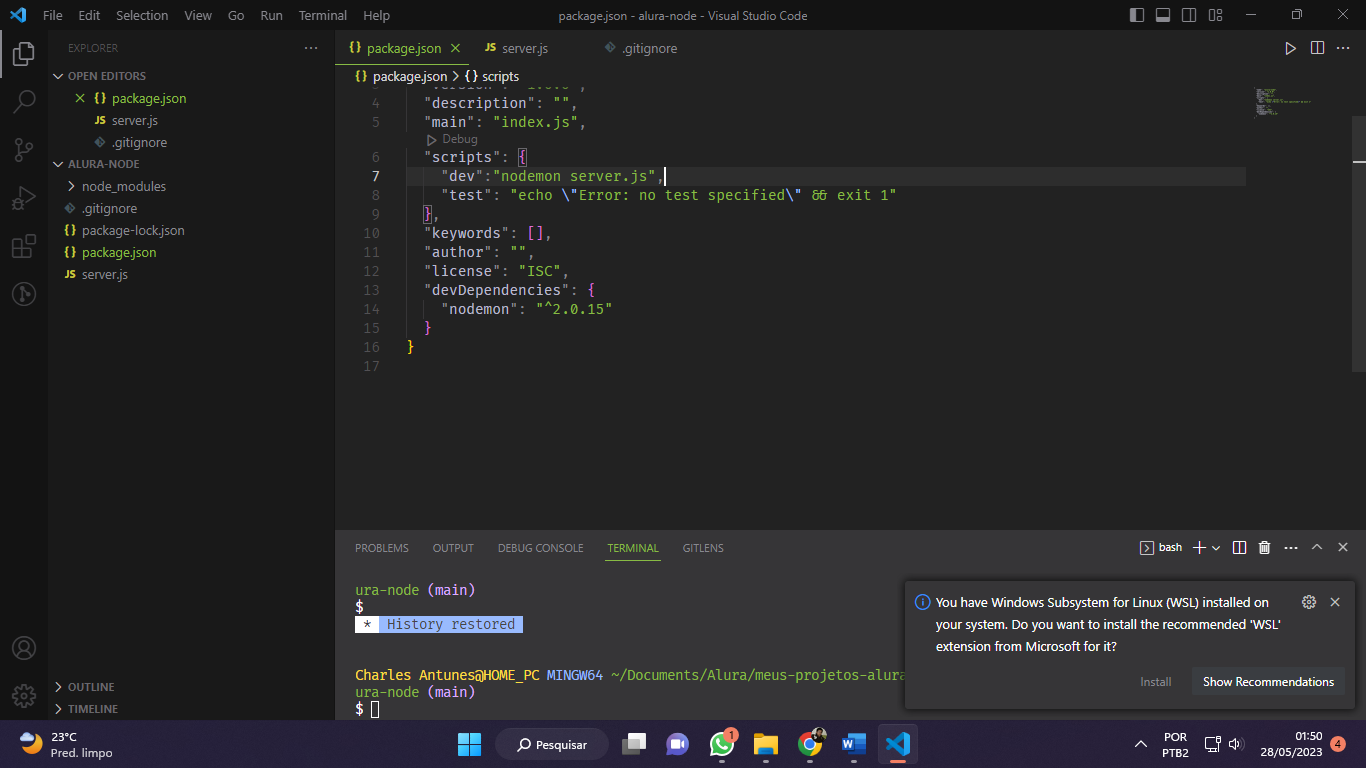
* Dependências para iniciar e salvar automaticamente o projeto, mostra se contém erros no projeto.

npm install [nodemon@2.0.15](mailto:nodemon@2.0.15) -D

obs. -D é para mostrar que é apenas usado no desenvolvimento e quando for para produção ele não vai, vai gerar a pasta node\_modules, onde todas as bibliotecas baixadas vão ser guardadas, por isso criamos a pasta/arquivo .gitignore e adicionamos dentro o node\_modules.



* Ativando o nodemon no package.json, adicionando na “scripts” uma linha com "dev":"nodemon server.js"



Obs. Para executar o nodemon, basta executar npm run dev onde o dev foi adicionado no scripts do package.json

========================================================================================================

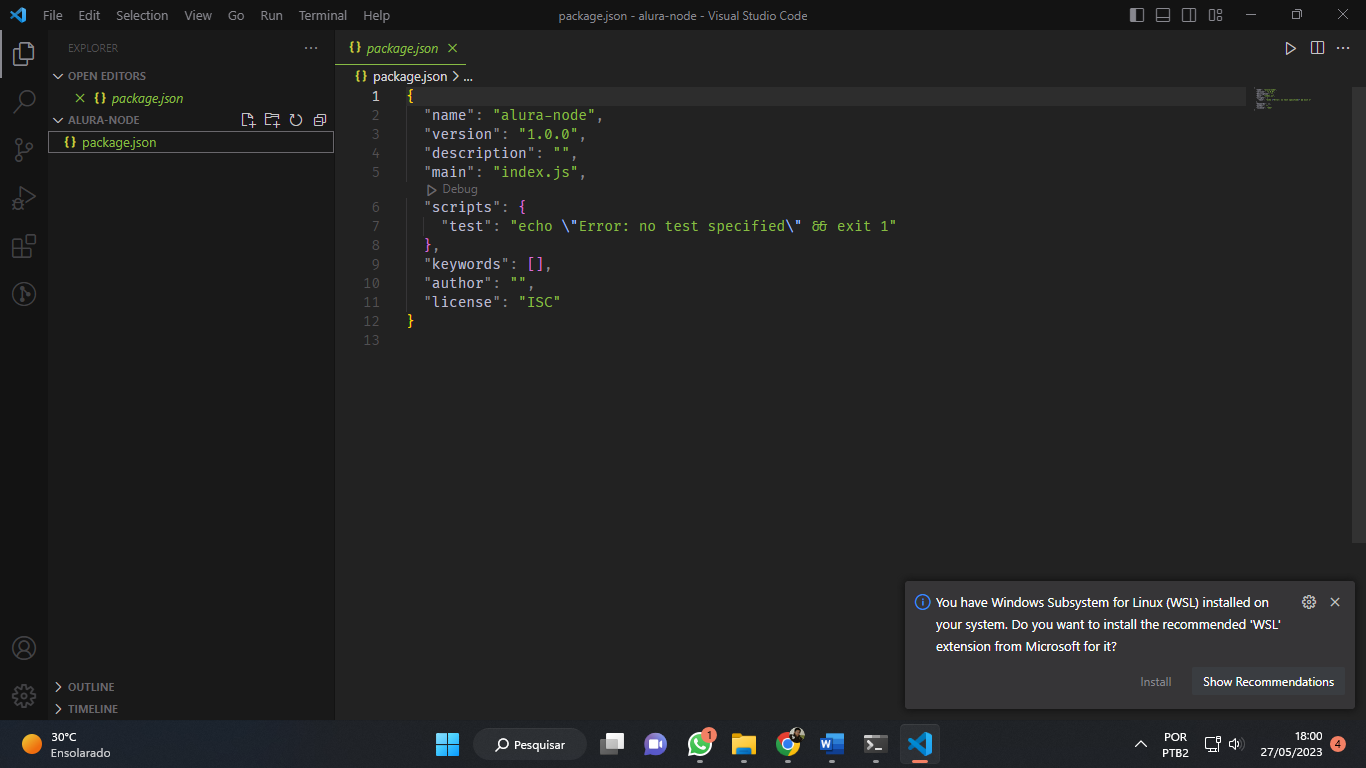
1 – No CMD

mkdir alura-node = criando a pastar do projeto

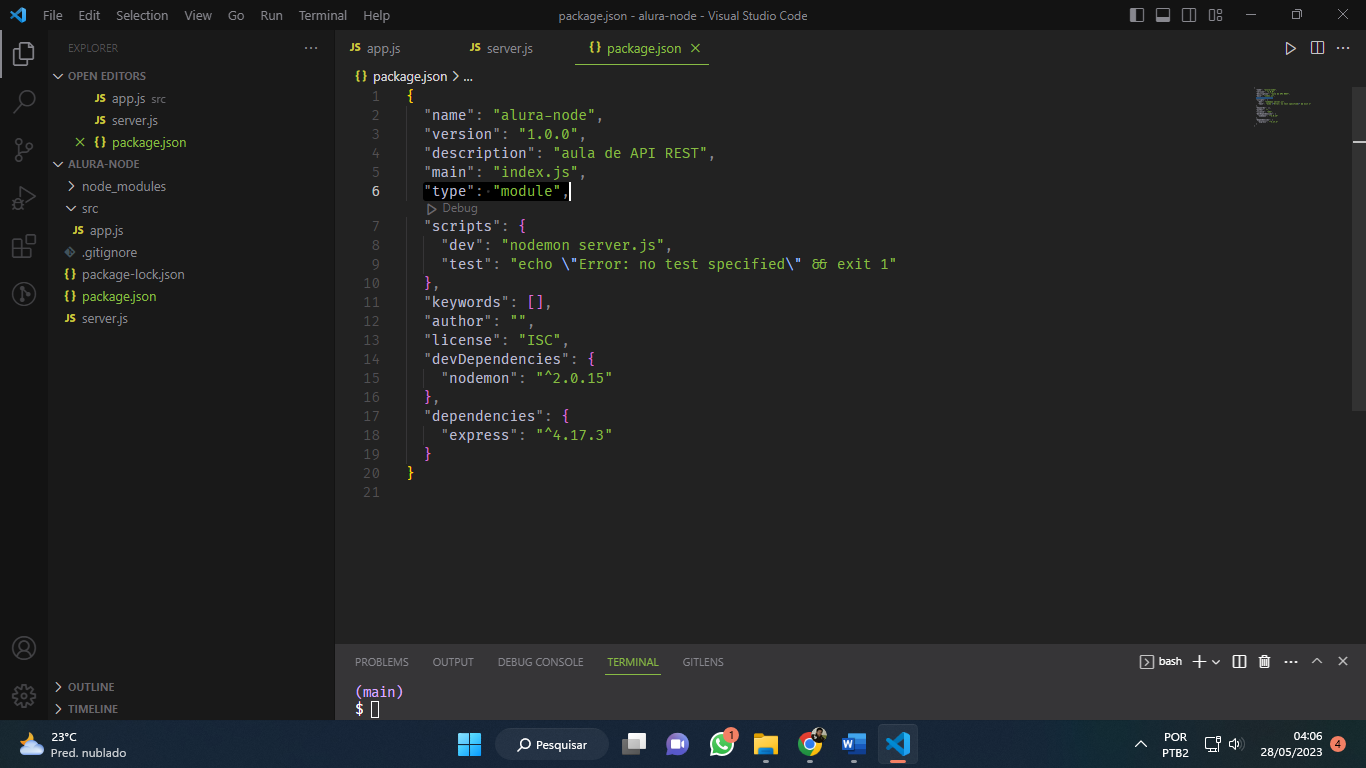
cs mkdir alura-node = entrando na pasta do projeto

npm init -y = criando o projeto, -y quer dizer que as perguntas de criação não aparecem, já vão ser respondida

code . = abrir o vcsCode



* No package.json, vamos cria informação “type”:”module”, para informar que vamos trabalhar com módulos.



O projeto vem com esse arquivo como principal para iniciar, ele usa o “main” como rota para o arquivo principal que é o index.js

==============================================================================================================

2 – Criando o servidor local, criar na raiz da pasta o arquivo server.js e importando os componentes nativos.

Dentro do **server.js**

*const* http = require('http');

*const* port = 3000;  *//porta da comunicação do servidor local, depois tem de mudar e criar uma variável de ambiente que irá definir a porta.*

*//criando o servidor*

*const* server = http.createServer((*req*,*res*)=> {

    res.writeHead(200, {'Content-type' : 'text/plain'}); *//resposta é um status escrito na cabeça da requisição(writeHead), o tipo de resposta é um texto.*

    res.end('Curso de Node') *//resposta no body do html, texto teste, depois será mudado*

})

*//mostrando a porta que está usando*

server.listen(port,()=>{

    console.log(**`**Servidor escutando em http://localhost:${port}**`**)

})

Terminal

$ node server.js

Servidor escutando em <http://localhost:3000>

Obs.: clicando no link, abre a página html com a palavra - Curso de Node

Ao instalar o Node, já temos à nossa disposição alguns módulos embarcados para começar a desenvolver nossas bibliotecas e aplicações. Para conhecer mais sobre esses recursos nativos, acesse a documentação oficial referente às APIs nesse [link](https://nodejs.org/api/).

=================================================================================================================

3 – Dentro do server.js, definindo as **rotas**, depois tem de separar as rotas em um arquivo separado.

Adicionado rotas e alteração na res.end().

*const* rotas = {

    '/': 'Curso de Nodejs',

    '/livros': 'Entrei na pag. de livros',

    '/autores': 'Listagem de autores'

}

*//criando o servidor*

*const* server = http.createServer((*req*,*res*)=> {

    res.writeHead(200, {'Content-type' : 'text/plain'}); *//resposta é um status escrito na cabeça da requisição(writeHead), o tipo de resp é um texto.*

    res.end(rotas[req.url]);

})

Terminal

$ node server.js

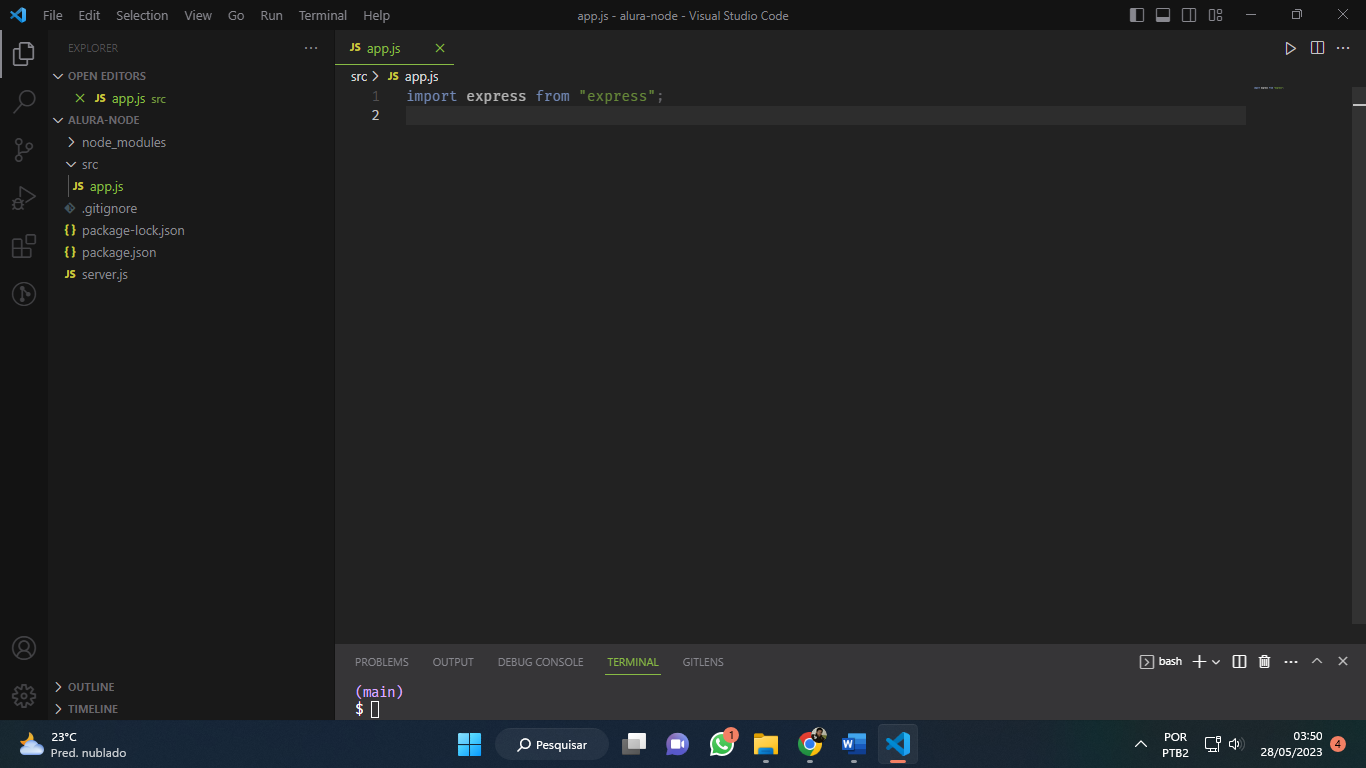
Servidor escutando em <http://localhost:3000>

Obs. ao abrir a página, será direcionado para a rota home(‘/’) com o texto - Curso de Nodejs, depois na página, altere o final da url as duas rotas declaradas, 1°= <http://localhost:3000/livros>, 2º= <http://localhost:3000/autores> . Será exibido -Entrei na pag. de livros e Listagem de autores.

4 – Express , instalando o framework/biblioteca

npm install [express@4.17.3](mailto:express@4.17.3)

* padronizando o projeto com pasta src e dentro, criando o arquivo app.js



* trabalhando no arquivo app.js

import express from "express";

*const* app = express();

*//simulando uma biblioteca para testar a visualizar dos livros na página.*

*const* livros = [

    {

        id: 1,

        "titulo" : "Senhor dos Aneis"

    },

    {

        id: 2,

        "titulo" : "O Hobit"

    }

]

*//rotas com as infos*

app.get('/', (*req*,*res*)=>{  *//buscar as inf da home*

    res.status(200).send('Curso de Nodejs com API'); *// resposta e msg*

})

app.get('/livros',(*req*,*res*)=>{

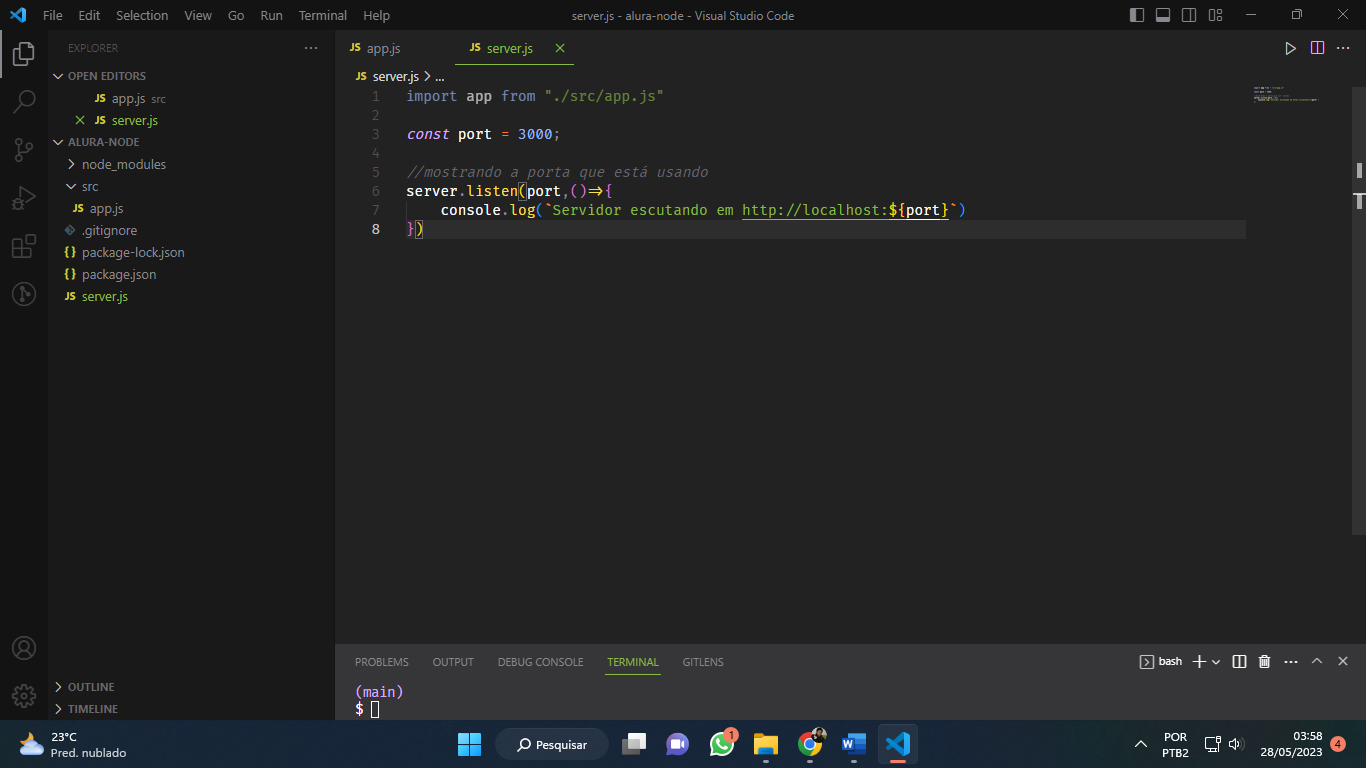
    res.status(200).json(livros) *//convertendo p json os arquivos recebidos*

})

export default app  *//exportando para esses dados serem trabalhados no server.js*

* Apagando o tipo de exportação que é o require e colocando o import, também apagando os códigos de teste de server.js.

Obs. Como o express foi importado no app.js, vamos importar apenas o app.js que já vem com o express. Não esquece de da extensão .js



Obs. Apaguei os comentários do port.

* Iniciando os novos códigos, com alteração na port, adicionado duas opções de portas. Outra foi apagando o server e colocando app

import app from "./src/app.js"

*const* port = process.env.PORT || 3000;

*//mostrando a porta que está usando*

app.listen(port,()=>{

    console.log(**`**Servidor escutando em http://localhost:${port}**`**)

})

* Inicializando para testar

npm run dev

[nodemon] 2.0.15

[nodemon] to restart at any time, enter **`**rs**`**

[nodemon] watching path(s): \*.\*

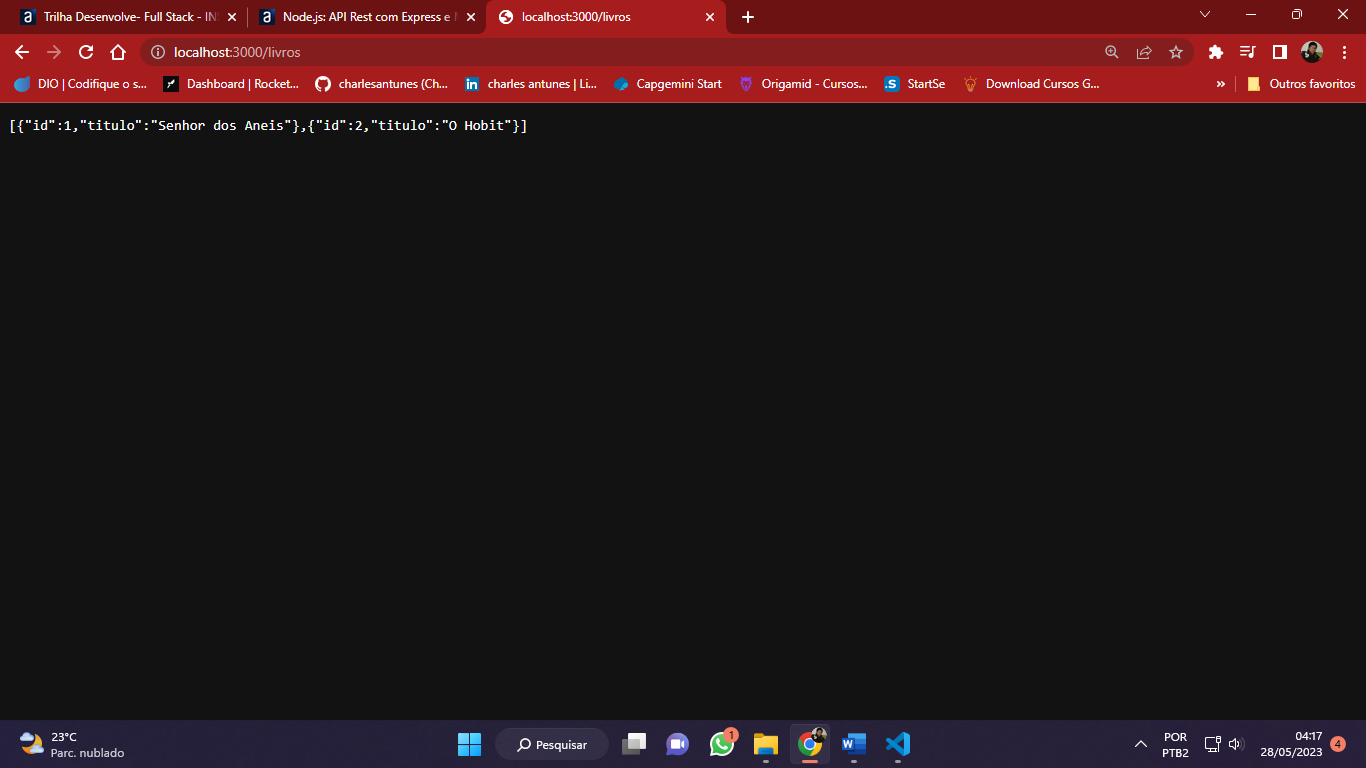
[nodemon] watching extensions: js,mjs,json

[nodemon] starting **`**node server.js**`**

Servidor escutando em http:*//localhost:3000*

Obs. Observe que estou usando nodemon e se não aparecer o “servidor escutando em http://localhost:3000 “, significa que tem erro no cod

* Na página, se alterar o final da url vai aparecer os livros com os filmes criados em json



========================= **INICIANDO O CRUD, TUDO JUNTO, SEM ESTRUTURA, APENAS TESTE** =========================

import express from "express";

*const* app = express();

app.use(express.json()) *//converte o entra como json() em objeto e ser tratado*

*//simulando uma biblioteca para testar a visualizar dos livros na página.*

*const* livros = [

    { id: 1, "titulo" : "Senhor dos Aneis" },

    { id: 2, "titulo" : "O Hobit" }

]

* **PESQUISAR TODOS OS LIVROS.**

*//===================== Pesquisa todos os Livros =====================================*

*//rotas com as infos*

app.get('/', (*req*,*res*)=>{  *//buscar as inf da home*

    res.status(200).send('Curso de Nodejs com API'); *// resposta e msg*

})

app.get('/livros',(*req*,*res*)=>{

    res.status(200).json(livros) *//convertendo p json os arquivos recebidos*

})

* **ADICIONAR**

*//====================== Adicionar ================================================*

app.post('/livros', (*req*,*res*)=>{

    livros.push(req.body)   *//push() = adicionar, o que estiver no corpo, adiciona em livros*

    res.status(201).send('O Livro foi cadastrado com sucesso')

})

export default app  *//exportando para esses dados serem trabalhados no server.js*

obs. Codigo organizado no array livros, adicionado o post e no começo adicionado add.use(express.json()) para converter json() em objeto.

* **ATUALIZAR**

*//====================== Atualizar ================================================*

app.put('/livros/:id', (*req*,*res*)=>{

*let* index = buscaLivros(req.params.id);*// req.params.id é o id do endereço*

    livros[index].titulo = req.body.titulo; *//atualiza o títudo do corpo*

    res.status(200).json(livros)

})

*function* buscaLivros(*id*) { *//func p/ buscar livro pelo id, depois ela é substituida pelo bd*

    return livros.findIndex(*Element* => Element.id == id)*//findIndex() = verifica todos indices*

}

* **PESQUISAR UM LIVRO (essa vai ser inserido no início, junto com o pesquisar todos)**

*//====================== Pesquisa um Livro============================================*

app.get('/livros/:id', (*req*,*res*)=>{

*let* index = buscaLivros(req.params.id);

    res.status(200).json(livros[index]);

})

========================================== **NO POSTMAN** ==========================================

* **PESQUISAR TODOS OS LIVROS.**

GET http://localhost:3000/livros

Resposta:

[

    {

        "id": 1,

        "titulo": "Senhor dos Aneis"

    },

    {

        "id": 2,

        "titulo": "O Hobit"

    }

]

* **PESQUISAR UM LIVRO**

GET http://localhost:3000/livros/

Resposta:

{

    "id": 2,

    "titulo": "O Hobit"

}

* **ADICIONAR**

POST http://localhost:3000/livros/

{

    "id": 3,

    "titulo": "Matrix"

}

Resposta:

O Livro foi cadastrado com sucesso

* **ATUALIZAR** (o id vai no URL, 3 era o Matriz)

PUT http://localhost:3000/livros/3

{

    "titulo": "Velozes e furiosos"

}

Resposta:

[

    {

        "id": 1,

        "titulo": "Senhor dos Aneis"

    },

    {

        "id": 2,

        "titulo": "O Hobit"

    },

    {

        "id": 3,

        "titulo": "Steve Jobs"

    }

]

DELETE http://localhost:3000/livros/1

Resposta:

Livro 1 removido com sucesso

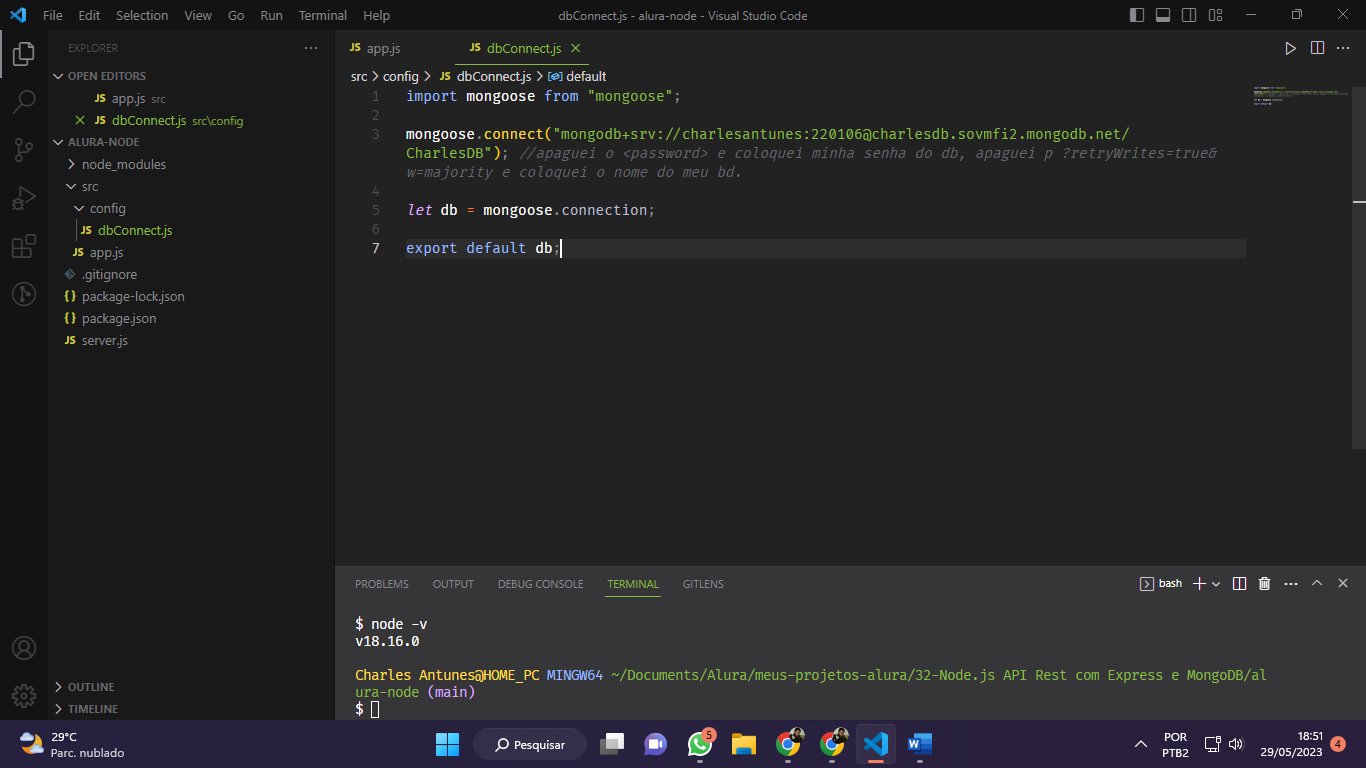
========================================== **FIM DO CRUD TESTE** =========================================

================= REALIZANDO AS MUDANÇAS PARA BOAS PRÁTICAS E COMUNICAR COM MONGODB CLOUD ======================

1. – Instalando o mongoose (auxilia na comunicação com o bd)

npm install mongoose@6.2.6

Dentro da pasta src, criar uma pasta com o nome config e dentro criar o arquivo dbConnect.js, buscando o link de conexão no mongodb/database/connect/drivers.



import mongoose from "mongoose";

mongoose.connect("mongodb+srv://charlesantunes:220106@charlesdb.sovmfi2.mongodb.net/CharlesDB"); *//apaguei o <password> e coloquei minha senha do db, apaguei p ?retryWrites=true&w=majority e coloquei o nome do meu DATABASE.*

*let* db = mongoose.connection;

export default db;

* No app.js, criada 3 linhas, import db, db.on e db.once.

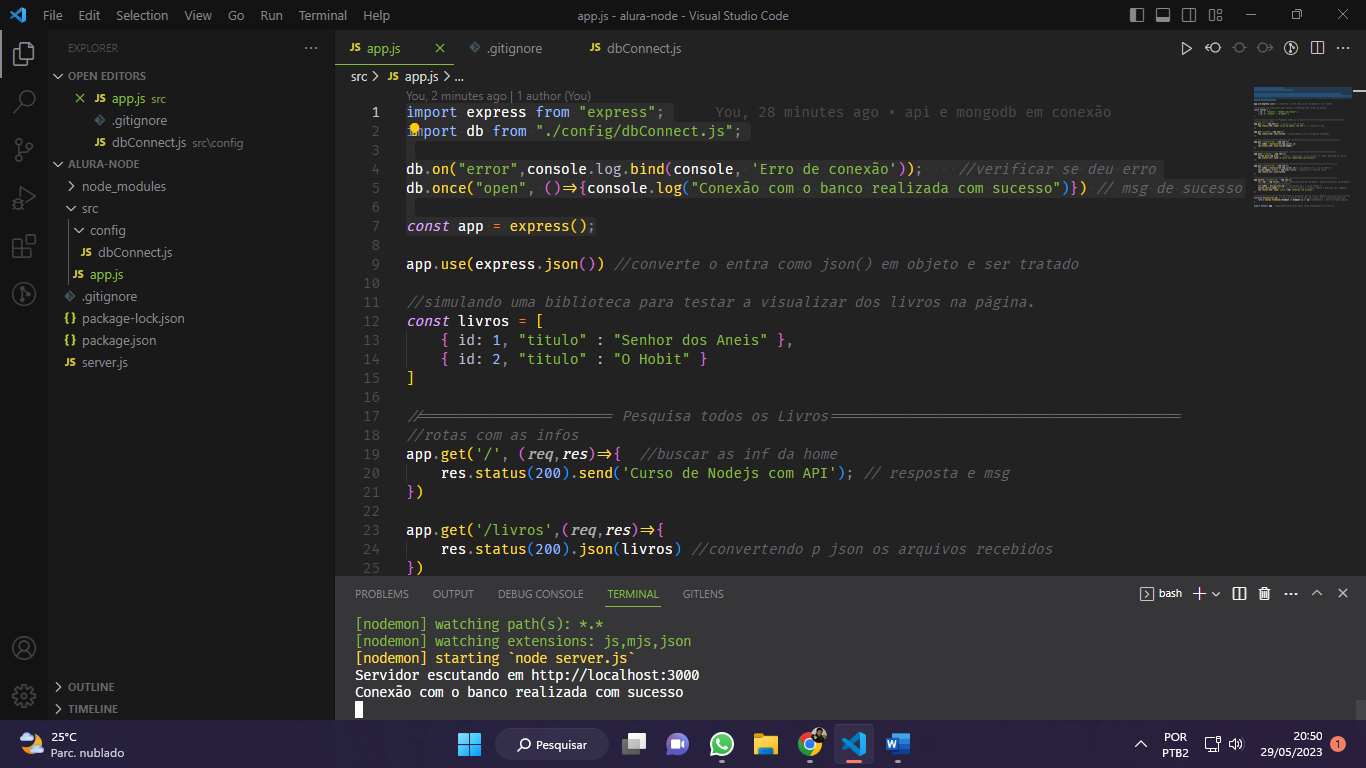
import express from "express";

import db from "./config/dbConnect.js";

db.on("error",console.log.bind(console, 'Erro de conexão'));    *//verificar se deu erro*

db.once("open", ()=>{console.log("Conexão com o banco realizada com sucesso")}) *// msg de sucesso*

*const* app = express();

iniciando o projeto npm run dev. Verifica que a msn de “Conexão com o banco realizada com sucesso”, aparece no terminal

Criando meu DATABASE (CharlesBD) e minha tabelas(livros).

Obs. id já é gerado automaticamente  
 **titulo**: "Lógica de programação"

**autor**: "Charles Antunes"

**editora**: "Casa do Código"

**numeroPaginas**: 150

========================================= Dica ======================================================

* **Liberar acesso a qualquer IP:**

Para liberar o acesso a qualquer IP via [Mongo Atlas](https://www.mongodb.com/cloud/atlas/). Após fazer o login no Mongo Atlas, acesse o seu cluster na dashboard do Atlas e siga os passos:

Network Access > IP Access List > Add an IP Address > allow access from everywhere> Confirm;

====================================================================================================

6– Em src, criar a pasta models (nela será incluso a regra de negócios, CRUD). Será criado dentro de models o arquivo Livros.js (L maiúsculo)

import mongoose from "mongoose";

*//esse é o esqueleto da tabela livros, quando adicionado algum livro, precisa desses parametros abaixo, se estiver required é obrigatorio ter para adicionar.*

*//obs. se o banco de dados não estiver sido criado no mongodb, com esses códigos ele vai ser criado automaticamnete.*

*const* livrosSchema = new mongoose.Schema(*//criação do banco de dados, Schema() são os paramentros e tipos*

    {

        id: {type: String}, *//id não precisa de required, pq já vem de padrão*

        titulo: {type: String},

        autor: {type: String, required: true},  *//required = se é obrigatório*

        editora: {type: String, required: true},  *//required = se é obrigatório*

        numeroPaginas: {type: Number}

    }

);

*const* livros = mongoose.model('livros', livrosSchema); *//livros = se não tiver essa Collection/tabela/coleção no bd, cria sozinho.*

export default livros;

* no app.js, importa a classe livros(vai ficar no app.js por em quanto essa classe, até estruturar o código), realizar alguma alterações app.get que consultada todos os livros(alteração para testar a comunicação)

app.get('/livros',(*req*,*res*)=>{

    livros.find((*err*,*livros*)=>{ *//find() pega tudo, err = se tiver erro vai devolver nesse parametro*

    res.status(200).json(livros)

    })

})

* apagando a const livros, ela foi substituída pela const livrosSchema do arquivo Livros.js

linha apagada, teste já foi feito.

*//simulando uma biblioteca para testar a visualizar dos livros na página.*

*const* livros = [

    { id: 1, "titulo" : "Senhor dos Aneis" },

    { id: 2, "titulo" : "O Hobit" }

]

* Testando a conexão get no postman

GET http://localhost:3000/livros

Resposta:

[

    {

        "\_id": "64754f26b43f626e973f7fe7",

        "titulo": "Lógica de programação",

        "autor": "Charles Antunes",

        "editora": "Casa do Código",

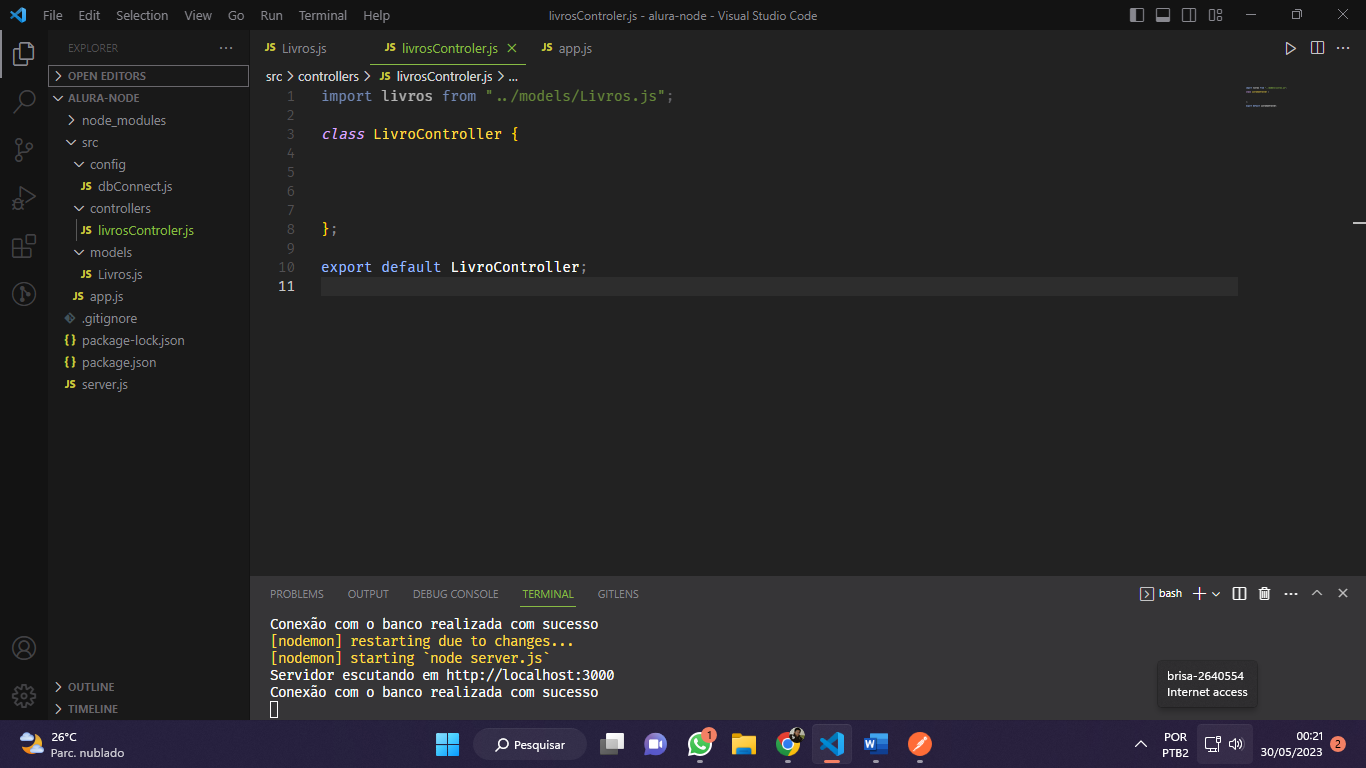
        "numeroPaginas": 150

    }

]

========================================================================================================

6 – Em src, criar a pasta **controllers**, nela serão colocados os arquivos CRUD, chamado de **VIEW do modelo MVC (Model-View-Controller)**, esses arquivos estão no app.js e vão para controller. Dentro será criado o arquivo livrosController.js. Dentro será criado uma Classe para colocar todo o CRUD dentro.



Criando os **métodos CRUD**, eles serão criados com o STATIC para não precisar ser estanciado. Vou substituir o app.get , app.post, app.put, app.delete, porque eles vão ser chamados nas rotas, logo, vamos usar rotas para gerencias onde vão ser chamados os verbos http: GET, POST, PUT e DELETE. Agora precisamos mostrar o destino dos métodos criados no lugar deles, isso será feito com todas, dessa forma precisamos colocar nomes intuitivos nossos, vou começar com o nome para método **listarLivros** que era **app.get** .

import livros from "../models/Livros.js";

*class* LivroController {

*static* listarLivros = (*req*,*resp*) =>{ //substituiu o app.get('/', (req,res)=>{

*//colocar os arquivos da função “Pesquisa todos os Livros” que está no app.js*

    }

};

export default LivroController;

* Em src, criar a pasta **routes**(rotas), dentro criar um arquivo chamado livrosRoutes.js. Aqui colocamos importamos o express() e colocamos os métodos: GET, POST, PUT e DELETE.

import express from "express";

import LivroController from "../controllers/livrosController.js";

*const* router = express.Router(); *//usando o Router() do express()*

router.

    get('/livros',LivroController.listarLivros);

export default router;  *//vai ser chamado no app.js*

* Em routes, criar o arquivo chamado index.js. Aqui importamos todas as rotas criadas na pasta, evitar no caso ter muitos arquivos de rotas e nos outros arquivos ter de importar todos, no caso vai importado só o index.js

import express from 'express';

import livros from './livrosRoutes.js'

*const* routes = (*app*) => {

    app.route('/').get((*req*,*res*)=>{ *//essa rota da home, será feita aqui por ser simples*

        res.status(200).send({titulo: "Curso node.js"})

    })

    app.use( *//aqui colocar todas as rotas*

        express.json(),

        livros

    )

}

export default routes; *// chamar no app.js*

* No app.js, importar o routes, e chamar o app de rotas dentro de routes que foi criado no importe de rotas. Excluir também a mensagem da home quando iniciada, porque ela foi criada em rotas.

import express from "express";

import db from "./config/dbConnect.js";

import livros from "./models/Livros.js";

import routes from "./routes/index.js"

db.on("error",console.log.bind(console, 'Erro de conexão'));    *//verificar se deu erro*

db.once("open", ()=>{console.log("Conexão com o banco realizada com sucesso")}) *// msg de sucesso*

*const* app = express();

app.use(express.json())

routes(app); *//recendo todas as rotas*

*//====================== Pesquisa todos os Livros=====================================*

*//rotas com as infos*

*// app.get('/', (req,res)=>{  //buscar as inf da home*

*//     res.status(200).send('Curso de Nodejs com API'); // resposta e msg*

*// })*

* Testando no postman as alterações.

Tabela livros.

GET http://localhost:3000/livros

Resposta:

[

    {

        "\_id": "64754f26b43f626e973f7fe7",

        "titulo": "Lógica de programação",

        "autor": "Charles Antunes",

        "editora": "Casa do Código",

        "numeroPaginas": 150

    }

]

Home da página.

GET http://localhost:3000

Resposta:

{

    "titulo": "Curso node.js"

}

=======================================================================================================

7 – Em livrosController.js, adicionar o método de **listarUmLivro.** (para organização, colocar em baixo do listarLivros)

*static* listarUmLivros = (*req*,*res*) =>{

*const* id = req.params.id;

        livros.findById(id,(*err*,*livros*)=>{

            if (err) {

                res.status(400).send({message:**`**${err.message} -Id do livro não encontrado}**`**}) *//400 erro do usuário*

            } else {

                res.status(200).send(livros);

            }

        })

    }

* Testando no postman a **listarUmLivro**.

Tabela livros.

GET http://localhost:3000/livros/6475937b7be34fc5aa692662

Obs. Precisa colocar o id que foi gerado pelo mongodb automático no bd

Resposta:

{

    "\_id": "6475937b7be34fc5aa692662",

    "titulo": "API com Node.js",

    "autor": "Charles Antunes",

    "editora": "Meu PC",

    "numeroPaginas": 40,

    "\_\_v": 0

}

8 – Em livrosController.js, adicionar o método de **adicionarLivro.**

*static* cadastrarLivros = (*req*,*res*) =>{

*const* livro = new livros(req.body); *//criando um novo objeto p/ armazenar os dados do corpo*

        livro.save((*err*)=>{

            if (err) {

                res.status(500).send({message:**`**${err.message} - Falha ao cadastrar livro.**`**})

            } else {

                res.status(201).send(livro.toJSON());

            }

        })

    }

* No arquivo livrosRoutes.js, adicionar a rota de cadastrarLivro (POST)

import express from "express";

import LivroController from "../controllers/livrosController.js";

*const* router = express.Router(); *//usando o Router() do express()*

router.

    get('/livros',LivroController.listarLivros);

    post('/livros', LivroController.cadastrarLivros)

export default router;  *//vai ser chamado no app.js*

* Testando no postman a **adição**.

Tabela livros.

POST http://localhost:3000/livros

Corpo da página(não precisa da id, já cria automático)

{

    "titulo": "API com Node.js",

    "autor": "Charles Antunes",

    "editora": "Minha casa",

    "numeroPaginas": 20

}

Resposta:

{

    "titulo": "API com Node.js",

    "autor": "Charles Antunes",

    "editora": "Minha casa",

    "numeroPaginas": 20,

    "\_id": "6475937b7be34fc5aa692662",

    "\_\_v": 0

}

Obs. “\_\_v”: 0 é o versionamento, indica quando teve modificação, é automático, não precisa mexer, pode tirar no momento que estiver criando a tabela tem a opção version key, colocar false.

======================================================================================================

9 – Em livrosController.js, adicionar o método de **AtualizarLivro.**

*static* atualizarLivros = (*req*,*res*) =>{

*const* id = req.params.id;

        livros.findByIdAndUpdate(id,{$set: req.body},(*err*)=>{  *//usar id para consulta e $set para adicionar*

            if (!err) {

                res.status(200).send({message: "O livro foi atualizado com sucesso"})

            } else {

                res.status(500).send({message: err.message})

            }

        })

}

* No arquivo livrosRoutes.js, adicionar a rota de **atualizarLivro** (PUT)

router

    .get('/livros',LivroController.listarLivros)

    .post('/livros', LivroController.cadastrarLivros)

    .put('/livros/:id', LivroController.atualizarLivros)

* Testando no postman a **edição/atualização**.

Tabela livros.

PUT http://localhost:3000/livros/6475937b7be34fc5aa692662

Obs. Precisa colocar o id que foi gerado pelo mongodb automático no bd

Corpo da requisição

{

    "editora": "Meu PC",

    "numeroPaginas": 40

}

Resposta:

"message": "O livro foi atualizado com sucesso"

======================================================================================================

9 – Em livrosController.js, adicionar o método de **excluirLivro.**

*static* excluirLivro = (*req*,*res*)=>{

*const* id = req.params.id;

        livros.findByIdAndDelete(id, (*err*) =>{

            if (!err) {

                res.status(200).send(**`**ID ${id} deletado com sucesso**`**)

            } else {

                res.status(400).send({message:err.message})

            }

        })

}

* No arquivo livrosRoutes.js, adicionar a rota de **excluirLivro** (DELETE)

router

    router

    .get('/livros',LivroController.listarLivros)

    .get('/livros/:id',LivroController.listarUmLivros)

    .post('/livros', LivroController.cadastrarLivros)

    .put('/livros/:id', LivroController.atualizarLivros)

    .delete('/livros/:id', LivroController.excluirLivro)

* Testando no postman a **exclusão**.

Tabela livros.

DELETE http://localhost:3000/livros/6475937b7be34fc5aa692662

Obs. Precisa colocar o id que foi gerado pelo mongodb automático no bd

Resposta:

{

    "message": "Livro removido com sucesso"

}

=======================================================================================================

* App.js antes das modificações

import express from "express";

import db from "./config/dbConnect.js";

import livros from "./models/Livros.js";

db.on("error",console.log.bind(console, 'Erro de conexão'));    *//verificar se deu erro*

db.once("open", ()=>{console.log("Conexão com o banco realizada com sucesso")}) *// msg de sucesso*

*const* app = express();

app.use(express.json()) *//converte o entra como json() em objeto e ser tratado*

*//====================== Pesquisa todos os Livros=========================================*

*//rotas com as infos*

app.get('/', (*req*,*res*)=>{  *//buscar as inf da home*

    res.status(200).send('Curso de Nodejs com API'); *// resposta e msg*

})

app.get('/livros',(*req*,*res*)=>{

    livros.find((*err*,*livros*) =>{ *//find() pega tudo, err = se tiver erro vai devolver nesse parametro*

    res.status(200).json(livros)

    })

})

*//====================== Pesquisa um Livro============================================*

app.get('/livros/:id', (*req*,*res*)=>{

*let* index = buscaLivros(req.params.id);

    res.status(200).json(livros[index]);

})

*//====================== Adicionar ================================================*

app.post('/livros', (*req*,*res*)=>{

    livros.push(req.body)   *//push() = adicionar, o que estiver no corpo, adiciona em livros*

    res.status(201).send('O Livro foi cadastrado com sucesso')

})

*//====================== Atualizar ================================================*

app.put('/livros/:id', (*req*,*res*)=>{

*let* index = buscaLivros(req.params.id);*// req.params.id é o id do endereço*

    livros[index].titulo = req.body.titulo; *//atualiza o títudo do corpo*

    res.status(200).json(livros)

})

*//====================== Deletar ================================================*

app.delete('/livros/:id', (*req*,*res*)=>{

*let* {id} = req.params; *//{id}= desestruturação por parametro, quando atribuimos um parametro a uma variável, vem entre chaves.*

*let* index = buscaLivros(id);*// req.params.id é o id do endereço*

    livros.splice(index,1)*//1º a posição do indice p/ apagar, depois a qtde que será apagado.*

    res.status(200).send(**`**Livro ${id} removido com sucesso**`**)

})

*//====================== Função usado no pesquisar por um livro e editar =======================*

*function* buscaLivros(*id*) { *//func p/ buscar livro pelo id, depois ela é substituida pelo bd*

    return livros.findIndex(*Element* => Element.id == id)*//findIndex() = verifica todos indices*

}

export default app;  *//exportando para esses dados serem trabalhados no server.js*

* App.js depois das modificações

import express from "express";

import db from "./config/dbConnect.js";

import livros from "./models/Livros.js";

import routes from "./routes/index.js"

db.on("error",console.log.bind(console, 'Erro de conexão'));    *//verificar se deu erro*

db.once("open", ()=>{console.log("Conexão com o banco realizada com sucesso")}) *// msg de sucesso*

*const* app = express();

app.use(express.json())

routes(app); *//recendo todas as rotas*

*//====================== Pesquisa todos os Livros=========================================*

*//rotas com as infos*

*// app.get('/', (req,res)=>{  //buscar as inf da home*

*//     res.status(200).send('Curso de Nodejs com API'); // resposta e msg*

*// })*

*// app.get('/livros',(req,res)=>{*

*//     livros.find((err,livros) =>{ //find() pega tudo, err = se tiver erro vai devolver nesse parametro*

*//     res.status(200).json(livros)*

*//     })*

*// })*

*//====================== Pesquisa um Livro============================================*

*// app.get('/livros/:id', (req,res)=>{*

*//     let index = buscaLivros(req.params.id);*

*//     res.status(200).json(livros[index]);*

*// })*

*//====================== Adicionar ================================================*

*// app.post('/livros', (req,res)=>{*

*//     livros.push(req.body)   //push() = adicionar, o que estiver no corpo, adiciona em livros*

*//     res.status(201).send('O Livro foi cadastrado com sucesso')*

*// })*

*//====================== Atualizar ================================================*

*// app.put('/livros/:id', (req,res)=>{*

*//     let index = buscaLivros(req.params.id);// req.params.id é o id do endereço*

*//     livros[index].titulo = req.body.titulo; //atualiza o títudo do corpo*

*//     res.status(201).json(livros)*

*// })*

*//====================== Deletar ================================================*

*// app.delete('/livros/:id', (req,res)=>{*

*//     let {id} = req.params; //{id}= desestruturação por parametro, quando atribuimos um parametro a uma variável, vem entre chaves.*

*//     let index = buscaLivros(id);// req.params.id é o id do endereço*

*//     livros.splice(index,1)//1º a posição do indice p/ apagar, depois a qtde que será apagado.*

*//     res.status(200).send(`Livro ${id} removido com sucesso`)*

*// })*

*//====================== Função usado no pesquisar por um livro e editar =======================*

*// function buscaLivros(id) { //func p/ buscar livro pelo id, depois ela é substituida pelo bd*

*//     return livros.findIndex(Element => Element.id == id)//findIndex() = verifica todos indices*

*// }*

export default app;  *//exportando para esses dados serem trabalhados no server.js*

* App.js atualmente e enxuto.

import express from "express";

import db from "./config/dbConnect.js";

import routes from "./routes/index.js"

db.on("error",console.log.bind(console, 'Erro de conexão'));

db.once("open", ()=>{console.log("Conexão com o banco realizada com sucesso")})

*const* app = express();

app.use(express.json())

routes(app);

export default app;

==================================== Criando relação entre tabelas===========================================

10 – Para criar relação entre tabelas, primeiro vamos fazer outra tabela, o mesmo esquema da tabela **livros.**

* Criar o arquivo **Autor.js** na pasta models

import mongoose from "mongoose";

*const* autorSchema = new mongoose.Schema(

    {

        id: {type: String},

        nome: {type: String, required: true},

        nacionalidade: {type: String}

    },

    {

        versionKey: false   *//criado automaticamente no mongodb, solicitei pra não criar*

    }

)

*const* autores = mongoose.model("autores", autorSchema);

export default autores;

* Criar o arquivo **autoresController.js** na pasta controllers.

import autores from "../models/Autor.js";

*class* AutorController {

*static* listarAutores = (*req*,*res*) =>{

        autores.find((*err*,*autores*) =>{

            res.status(200).json(autores)

        })

    }

*static* listarUmAutor = (*req*,*res*) =>{

*const* id = req.params.id;

        autores.findById(id,(*err*,*autores*)=>{

            if (err) {

                res.status(400).send({message:**`**${err.message} -Id do autor não encontrado}**`**}) *//400 erro do usuário*

            } else {

                res.status(200).send(autores);

            }

        })

    }

*static* cadastrarAutor = (*req*,*res*) =>{

*let* autor = new autores(req.body); *//criando um novo objeto p/ armazenar os dados do corpo*

        autor.save((*err*)=>{

            if (err) {

                res.status(500).send({message:**`**${err.message} - Falha ao cadastrar autor.**`**})

            } else {

                res.status(201).send(autor.toJSON());

            }

        })

    }

*static* atualizarAutores = (*req*,*res*) =>{

*const* id = req.params.id;

        autores.findByIdAndUpdate(id,{$set: req.body},(*err*)=>{  *//usar id para consulta e $set para adicionar*

            if (!err) {

                res.status(200).send({message: "O autor foi atualizado com sucesso"})

            } else {

                res.status(500).send({message: err.message})

            }

        })

    }

*static* excluirAutor = (*req*,*res*)=>{

*const* id = req.params.id;

        autores.findByIdAndDelete(id, (*err*) =>{

            if (!err) {

                res.status(200).send({message: **`**Autor removido com sucesso**`**})

            } else {

                res.status(500).send({message:err.message})

            }

        })

    }

};

export default AutorController

* Criar o arquivo **autoresRoutes.js** na pasta routes.

import express from "express";

import AutorController from "../controllers/autoresController.js";

*const* router = express.Router(); *//usando o Router() do express()*

router

    .get("/autores", AutorController.listarAutores)

    .get("/autores/:id", AutorController.listarUmAutor)

    .post("/autores", AutorController.cadastrarAutor)

    .put("/autores/:id", AutorController.atualizarAutores)

    .delete("/autores/:id", AutorController.excluirAutor)

export default router;

* Na pasta **index.js,** adicionar a rota de autores**.**

import express from 'express'

import livros from "./livrosRoutes.js";

import autores from "./autoresRoutes.js";

*const* routes = (*app*) => {

    app.route('/').get((*req*,*res*)=>{ *//essa rota da home, será feita aqui por ser simples*

        res.status(200).send({titulo: "Curso node.js"})

    })

    app.use( *//aqui colocar todas as rotas*

        express.json(),

        livros,

        autores

    )

}

export default routes;

* No postman, fazer os testes de inserção, consulta, atualização e delete. Verificar se tem as alterações no Cloud mongodb.

11 – Adicionar a referência de chave estrangeira no arquivo **Livro.js, propriedade autor.**

import mongoose from "mongoose";

*const* livrosSchema = new mongoose.Schema(

    {

        id: {type: String},

        titulo: {type: String},

        autor: {type: mongoose.Schema.Types.ObjectId,ref: 'autores', required: true},  *//ref de autores da pasta models,*

        editora: {type: String, required: true},

        numeroPaginas: {type: Number}

    }

);

*const* livros = mongoose.model('livros', livrosSchema);

export default livros;

* Populando o campo **autor** do banco de dados quando forem usados os métodos do arquivo **livrosControllers.js**. adicionando em cada método a função population() e exec().
* Adicionando population() e exec() no método **listarLivros.**

import livros from "../models/Livros.js";

*class* LivroController {

*static* listarLivros = (*req*,*res*) =>{

        livros.find()

        .populate('autor')  *//consulta Autor.js, mostra as inf de autor da tabela autores.*

        .exec((*err*,*livros*) =>{

        res.status(200).json(livros)

        })

    }

* Precisa alterar o tipo do autor no banco de dados, porque ele vai virar um objeto guardando id, nome, nacionalidade. Para fazer isso, no postman, basta copiar o id do autor que estiver adicionado no banco de dados e fazer um PUT (update) no livro que no banco de dados e no campo autor colar o id do autor. Fazendo isso o parâmetro autor da tabela livros, vai virar um objeto.

GET http://localhost:3000/livros

[

    {

        "\_id": "64754f26b43f626e973f7fe7",

        "titulo": "Lógica de programação",

        "editora": "Casa do Código",

        "numeroPaginas": 150

    }

]

Obs. Copiar apenas o id do livro

PUT http://localhost:3000/livros/64754f26b43f626e973f7fe7

Obs. PUT(atualizar) será usado o id do livro no campo de pesquisa.

 {

       "autor": "648266e5f0306ac13dc7671e"

  }

Obs. Colar o ID do autor, para facilitar, pegar esse id direto da tabela do mondobd .

* Fazendo uma pesquisa no Postman para ver o campo autor como objeto.

GET http://localhost:3000/livros

[

    {

        "\_id": "64754f26b43f626e973f7fe7",

        "titulo": "Lógica de programação",

        "autor": {

            "\_id": "648266e5f0306ac13dc7671e",

            "nome": "Charles Antunes",

            "nacionalidade": "brasileiro"

        },

        "editora": "Casa do Código",

        "numeroPaginas": 150

    }

]

========================================================================================================

1. – Adicionando population() e exec() no método **listarLivroPorId.**
2. *static* listarLivroPorId = (*req*,*res*) =>{
3. *const* id = req.params.id;
4. livros.findById(id)
5. .populate('autor','nome') *// adicionando o campo nome para especificar o campo único da pesquisa.*
6. .exec((*err*,*livros*)=>{
7. if (err) {
8. res.status(400).send({message:**`**${err.message} -Id do livro não encontrado}**`**}) *//400 erro do usuário*
9. } else {
10. res.status(200).send(livros);
11. }
12. })
13. }

* Fazendo uma pesquisa por id de livros no Postman para ver o **nome** do campo autor da tabela livros.

GET http://localhost:3000/livros/64754f26b43f626e973f7fe7

{

    "\_id": "64754f26b43f626e973f7fe7",

    "titulo": "Lógica de programação",

    "autor": {

        "\_id": "648266e5f0306ac13dc7671e",

        "nome": "Charles Antunes"

    },

    "editora": "Casa do Código",

    "numeroPaginas": 150

}

Obs. Veio o parâmetro autor, apenas com nome, id é padrão vim.

=============================== Realizar pesquisas por campos diferentes =================================

* Pesquisar livro por editora.

No arquivo **livrosControllers.js**. Adicionar um método no final, chamado **listarLivrosPorEditora.**

*static* listarLivroPorEditora = (*req*,*res*)=>{

*const* editora = req.query.editora; *//busca na página a palavra editora e passa por parametro na url.*

*//find() = aqui vai ser feito um filtro na página pelo campo editora e atribuir a variavel editora, depois comparar com o valor do campo editora do banco de dados.*

       livros.find({'editora':editora},{},(*err*,*livros*)=>{

        res.status(200).send(livros);

*//aqui pode incluir o if else do err*

       })

    }

No arquivo **livrosRoutes.js**. Adicionar a rota na frente do id, pra ser iniciada primeiro, isso é importante porque o id na frente vai dar erro na pesquisa do listarLivrosPorEditora.

import express from "express";

import LivroController from "../controllers/livrosController.js";

*const* router = express.Router(); *//usando o Router() do express()*

router

    .get('/livros',LivroController.listarLivros)

    .get('/livros/busca', LivroController.listarLivroPorEditora) *//buscar por editora*

    .get('/livros/:id',LivroController.listarLivroPorId)

    .post('/livros', LivroController.cadastrarLivros)

    .put('/livros/:id', LivroController.atualizarLivros)

    .delete('/livros/:id', LivroController.excluirLivro)

export default router;

* Testando no postman a busca pela editora

GET http://localhost:3000/livros/busca?editora=Casa do Código

Obs. Observe que usamos na url as palavras usadas na rota “busca”

Resposta

[

    {

        "\_id": "64754f26b43f626e973f7fe7",

        "titulo": "Lógica de programação",

        "autor": "648266e5f0306ac13dc7671e",

        "editora": "Casa do Código",

        "numeroPaginas": 150

    }

]

===================================== FIM =====================================

Bônus de mais pesquisas por outros campos

# req.query e req.params

Os query params, ou parâmetros de consulta, são um conjunto definido de parâmetros anexados ao final de uma URL. Os query params são aquelas extensões da URL que ficam após o '?' e ajudam a definir um **conteúdo** ou **ações** com base nos dados passados.

Para adicionar vários parâmetros, um '&' é adicionado entre cada um. Eles podem ser criados por qualquer variação de tipos ou comprimentos de objetos, como String, Arrays e Numbers. Segue um exemplo:

Vamos supor que você construa uma requisição com req.query da seguinte maneira

app.get('/users', (req, res) => {

const nomeDoUsuario = req.query.nome;

res.json({ nome: `${nomeDoUsuario} `});

});COPIAR CÓDIGO

A forma de consultar a informação via get será conforme abaixo:

[http://localhost:3000/users?nome=[nome]](http://localhost:3000/users?nome=%5Bnome%5D)

O que está entre os colchetes [] são os query.params, por exemplo, <http://localhost:3000/users?nome=jacqueline>

Nesse caso, ele acessou o endpoint **users** e passou um conteúdo para um parâmetro denominado **nome** e o método app.get atribuiu esse valor a uma variável nomeDoUsuario e está devolvendo esse conteúdo em formato json.

Por outro lado, o req.params do framework express, que estamos utilizando no curso, funciona como se fosse um parâmetro para **rotas**. Você irá realizar as requisições diretamente por uma rota e não apenas fazer uma consulta.

Exemplo:

vamos supor que eu queira acessar um livro com o id 1 via req.params

app.get('livros/:id', (req,res) => {

const id = req.params.id;

res.json({ id: `${id}`});

});COPIAR CÓDIGO

A forma de consultar essa informação seria pelo endpoint: <http://localhost:3000/livros/1>

Diferentemente do primeiro exemplo, onde utilizamos req.query, aqui nesse exemplo estamos utilizando req.params.

Repare que a diferença entre ambos é que o req.query é utilizado quando o conteúdo passado servirá para fazer uma consulta ou para alguma finalidade (como a do exemplo mesmo, que devolvemos na resposta o nome que foi passado na própria URL), já o req.params é utilizado quando o parâmetro é um segmento da URL que vai compor a rota.

Para isso, na definição de rotas utilizamos a identificação **:id** para indicar que o parâmetro que virá terá o nome de id, conforme o exemplo abaixo:

router.get("/livros/:id", LivroController.listarLivroPorId)