**Tutorial**: Nome do tutorial

**Versão:** xx.xx

**Objetivo**: objetivos a ser atingidos

**Pré-Requisitos:**

Compreender o que é MVC e MVC Full

# CRIAR UM SERVIDOR USANDO COM NODE E EXPRESS

1. Iniciar o NPM e o Express

npm init –y

npm install express

PS D:\Projetos\app\_express> npm init -y

Wrote to D:\Projetos\app\_express\package.json:

"name": "app\_express",

"version": "1.0.0",

"description": "",

"main": "app.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [],

"author": "",

"license": "ISC"

}

PS D:\Projetos\app\_express> npm install express

npm notice created a lockfile as package-lock.json. You should commit this file.

npm WARN app\_express@1.0.0 No description

npm WARN app\_express@1.0.0 No repository field.

+ express@4.18.2

added 58 packages from 42 contributors and audited 58 packages in 12.133s

8 packages are looking for funding

run `npm fund` for details

found 0 vulnerabilities

1. Confirmar se o express foi instalado

pm ls --depth=0

app\_express@1.0.0 D:\Projetos\app\_express

`-- express@4.18.2

1. Criar o arquivo app.js e Importar o Express

Dados do arquivo app.js

const express = require('express');

const app = express();

1. Compreender o que são rotas, requisições, respostas, modelo cliente servidor
2. Criar a rota / (raiz)

Alterar app.js

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

    res.send('Chegamos ate aqui');

})

1. Acrescentar as linhas que roda o servidor e geram log

const porta = 3000

app.listen(porta,()=>{

    console.log('Servidor Executando')

    console.log('Porta: ',porta)

    console.log('URL: ','http://localhost:'+porta)

})

1. Rodar o servidor a partir da linha de comando

Node app.js

Obs.: Servidor para com CTRL+C

1. Abrir a url do servidor
2. Criar uma nova rota

Parar o servidor

Criar a rota2 abaixo da rota raiz

app.get('/rota2',(req, res) => {

    res.send('rota2');

})

Rodar novamente o servidor

Confirmar se a rota2 está funcionando

http://localhost:3000/rota2

1. Tentar uma rota que não está configurada para ver o erro
2. Configurar uma nova rota enviando um form.

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

    res.send(`

     <form action="/form" method="POST">

        <label>Teste: </label>

        <input type="text">

        <input type="submit" value="enviar">

     </form>

    `

    );

})

Observe o erro ao clicar em enviar.

Não existem rotas cadastradas /form que usam post

1. Acrescentar a rota /form usando POST

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

    res.send('rota /form usando POST acessada com sucesso');

})

Use a aba network para confirmar dados sendo enviado via POST

Digite algum dado no formulário e observe o resultado em network ao clicar em enviar

Observe o request method

# USAR O NODEMON PARA NÃO PRECISAR REINICIAR O SERVIDOR

1. Instalar o nodemon como uma dependência de desenvolvimento

npm install nodemon --save-dev

D:\Projetos\app\_express> npm install nodemon --save-dev

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@~2.3.2 (node\_modules\chokidar\node\_modules\fsevents):

npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

npm WARN app\_express@1.0.0 No description

npm WARN app\_express@1.0.0 No repository field.

+ nodemon@2.0.22

added 32 packages from 28 contributors and audited 91 packages in 9.508s

11 packages are looking for funding

run `npm fund` for details

found 0 vulnerabilities

1. Rodar o projeto com nodemon diretamente

Executar o comando

npx nodemon app.js

PS D:\Projetos\app\_express> npx nodemon app.js

[nodemon] 2.0.22

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

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

[nodemon] watching extensions: js,mjs,json

[nodemon] starting `node app.js`

Servidor Executando

Porta: 3000

URL: <http://localhost:3000>

Modificar a linha da rota raiz

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

    res.send('Chegamos ate aqui e rodamos o projeto com nodemon');

})

Observe que o nodemon recarrega o projeto automaticamente

Abra a rota raiz para observar o resultado

1. Modificar o package.json para rodar nodemon de maneira mais amigável

  "main": "app.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "start": "nodemon app.js"

  },

Parar e rodar o servidor com npm

Npm start

# Criar uma rota com recebimento de parâmetros

1. Adicionar a seguinte rota:

app.get('/comparametros/:nome',(req, res) => {

    res.send('rota com parametros');

    console.log(req.params);

})

Efetuar o teste com parametros

<http://localhost:3000/comparametros/josue>

Observe a console do node

:nome faz com que o server espere um parâmetro após a última barra que vai ser chamados de nome

Req.params é um objeto que contém todos os parâmetros enviados

1. Modifique a rota para enviar o nome para console

app.get('/comparametros/:nome',(req, res) => {

    res.send('rota com parametros - '+req.params.nome);

    // console.log(req.params);

})

Obs.: neste ponto não é possível enviar mais quem send()

Obs2.:Também não é possível usar usar a rota sem parametros

Erro: Cannot GET /comparametros/

1. Torne o parâmetro opcional usando ?

Adicione um sinal de ? após o nome do parâmetro

app.get('/comparametros/:nome?',(req, res) => {

1. Adicione um segundo parâmetro

app.get('/comparametros/:nome?/:sobrenome?',(req, res) => {

    res.send('rota com parametros - '+req.params.nome);

    console.log(req.params)

})

Teste com

<http://localhost:3000/comparametros/josue/rosario>

1. Encaminhe todos os parâmetros diretamente para usuário como JSON

app.get('/comparametros/:nome?/:sobrenome?',(req, res) => {

    res.send(req.params);

    console.log(req.params)

})

# Modificando a rota para receber dados através de Query Strings

1. Adicione a rota:

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

    console.log(req.query)

    res.send(req.params);

})

Teste com

<http://localhost:3000/comquery?nome=josue&sobrenome=rosario>

obs.: Neste ponto observe que o objeto query e params são diferentes. Nada será enviado através da página

obs2: Existem problemas de segurança usar envio através de query

# Configurando e tratando envio de formulários através de POST

POSTS de formulários são tratados através do objetos req.body

1. Habilite o uso de req.body

Acrescente estas linha no início do arquivo após chamar o express

const app = express();

app.use(

    express.urlencoded(

        {

            extended:true

        }

    )

)

1. Modifique a rota post para mostrar o que está sendo enviado no body

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

    console.log(req.body)

    res.send('rota /form usando POST acessada com sucesso');

})

Atentar para o uso de req em vez de res uma vez que está sendo enviado uma requisição e está sendo tratada pelo servidor

Efetuar o teste preenchendo o campo e enviado através do botão. Um objeto vazio é mostrado

Servidor Executando

Porta: 3000

URL: http://localhost:3000

{}

Isto acontece porque o campo não possui o atributo name

1. Modifique a rota form e acrescente um name ao campo

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

    res.send(`

     <form action="/form" method="POST">

        <label for="campo\_teste">Teste: </label>

        <input type="text" name="campo\_teste">

        <input type="submit" value="enviar">

     </form>

    `

    );

})

Efetue o teste novamente e observe o envio do objeto req.body preenchido

# Configurando o Express para usar Routes com o padrão MVCFull

1. Crie o arquivo routes.js com as linhas

const express = require('express');

const route = express.Router();

1. Mova a rota /form para routes.js e altere como indicado.

route.get('/form',(req, res) => {

    res.send(`

     <form action="/form" method="POST">

        <label for="campo\_teste">Teste: </label>

        <input type="text" name="campo\_teste">

        <input type="submit" value="enviar">

     </form>

    `

    );

})

A partir desse momento as rotas usarão o objeto route

A função deste arquivo não é colocar o código das rotas em si mas apenas configurar.

Este arquivo carrega as rotas mas o código em si será responsabilidade do controller

Teste a rota /form para verificar se está funcionando corretamente

1. Criando a pasta e arquivo controllers/formController.js

Crie a pasta controllers com o arquivo formController.js dentro

exports.formView = (req,res) => {

};

formController é um modulo e formView é uma função que é exportada para ser usada no arquivo routes.js

1. Modifique a formView para enviar o form para o usuário.

Mova o res.send de routes.js para formController.js

exports.formView = (req,res) => {

    res.send(`

    <form action="/form" method="POST">

       <label for="campo\_teste">Teste: </label>

       <input type="text" name="campo\_teste">

       <input type="submit" value="enviar">

    </form>

   `

   );

};

1. Altere o route.js para importar formController e exportar todas as rotas que serão criadas

const express = require('express');

const route = express.Router();

const formController = require('./controllers/formController');

route.get('/form',formController.formView)

module.exports = route;

observe como ficou a route.get. Em vez de mostrar diretamente o form é chamada uma função de controller

A linha module.exports exporta todas a rotas para serem importadas por app.js

Modifique app.js

Acrescente a linhas para importar ao arquivo routes.js e também o importe o objeto routes.. O início do arquivo fica:

const express = require('express');

const app = express();

const routes = require('./routes')

app.use(

    express.urlencoded(

        {

            extended:true

        }

    )

)

app.use(routes)

**(...)**

Acesse a rota form para verificar se tudo está ok

1. Configure a rota de post para funcionar como as outras rotas

Routes.js receceberá uma nova rota

route.post('/comparametros/:nome?/:sobrenome?',formController.formPost)

formController.js receberá uma nova funcao a ser exportada

exports.formPost = (req,res) =>{

    res.send(req.params);

    console.log(req.params)

}

Em app.js a rota app.post('/form',(req, res) deixará de existir, portanto apague ou comente estas linhas

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

    console.log(req.body)

    res.send('rota /form usando POST acessada com sucesso');

})

Confirme se tudo continua funcionando, acesse a rota /form, escreva e envie conteúdo.

1. Configure a rota de home (/) para funcionar usando o routes

Routes.js receceberá um novo import e uma nova rota

const homeController = require('./controllers/homeController');

**(...)**

route.get('/',homeController.homeView)

Crie o arquivo Controllers/homeController.js

exports.homeView = (req,res) =>{

    res.send('Chegamos ate aqui e rodamos o projeto com nodemon');

}

Apague a rota raiz (/) de app.js

Confirme se a rota raiz está rodando

1. Mova as rotas rota2 e comquery para routes.js. Altere o objeto app por routes, teste e comente para manter o histórico de alterações e servir para futuras revisões

# Configurando o EJS

O processo de criação de views fica simples usando uma template engine que permite usar códigos javascript junto com o HTML para renderizar arquivos de templates.

Link: <https://ejs.co/>

1. Instale o ejs

npm i ejs

1. Configurando o EJS

.\app.js

//configuração do ejs

var path = require('path');

app.use(express.static(path.resolve(\_\_dirname,'public')))

app.set('view engine', 'ejs');

app.set('views',path.join(\_\_dirname+'/views/'));

Crie o arquivo .\views\home.ejs

<!DOCTYPE html>

<html lang="pt-BR">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>View Home</title>

</head>

<body>

    <h1> Renderizando ok</h1>

</body>

</html>

Modifique a função homeView() dentro de .\controllers\homeController.js para renderizar a view home.ejs

exports.homeView = (req,res) =>{

    //res.send('homeController, homeview()');

    console.log('homeController homeView()')

    res.render('home')

}

1. Modifique formController.formView

No modelo MVC as páginas de interação com o usuário são views. Portanto o formulário de cadastro também é uma view.

O conteúdo de formController.formView deverá virar o arquivo views\formulario.ejs e formView chama este arquivo.

controllers\formController.js

exports.formView = (req,res) => {

    // res.send(`

    //  <form action="/form" method="POST">

    //     <label>Teste: </label>

    //     <input name='teste' type="text">

    //     <input type="submit" value="enviar">

    //  </form>

    // `

    // )

    res.render('formulario')

}

views\formulario.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Formulario</title>

</head>

<body>

    <form action="/form" method="POST">

        <label>Teste: </label>

        <input name='teste' type="text">

        <input type="submit" value="enviar">

     </form>

</body>

</html>

1. Criando mais uma rota chamada cadastro com a respectiva view

Em .\routes.js cadastre a rota

route.get('/cadastro',formController.cadastro)

Em controllers\formController.js crie a função de cadastro

exports.cadastro = (req,res) =>{

    //res.send('homeController, homeview()');

    console.log('homeController cadastro()')

    res.render('cadastro',dados)

}

Crie o arquivo cadastro.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Cadastro</title>

</head>

<body>

    <h1>Cadastro</h1>

</body>

</html>

1. Configurando e criando um partial para a linha de Copyright

Um partial é um código que se repete em várias views. Para evitar duplicação de código ele é criado em arquivo separado e incluido na view, no local onde o código deveria se repetir.

Crie a pasta views\partials

Crie o arquivo views\partials\copy.ejs

<p>&copy; Copyright 2024 Propriedade de josue corporations</p>

Inclua a copy.ejs em views\formulario.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Formulario</title>

</head>

<body>

    <form action="/form" method="POST">

        <label>Teste: </label>

        <input name='teste' type="text">

        <input type="submit" value="enviar">

     </form>

     <%- include('./partials/copy.ejs') %>

</body>

</html>

1. Passando um array de dados para uma view

Crie o array dados e o objeto literal pessoa formController, na função cadastro

exports.cadastro = (req,res) =>{

    //res.send('homeController, homeview()');

    console.log('homeController cadastro()')

    dados = ['dado1','dado2','dado3','dado4']

    pessoa = {

        nome: 'josue micaroni',

        idade: '45'

    }

    res.render('cadastro',dados)

}

Obs.: Não é função do controller obter e tratar dados, mas por hora estamos enviando os mesmos prontos para a view. A função de obter dados é do model.

Modifique o body de cadastro.ejs para mostrar uma variável

<body>

    <h1>Cadastro</h1>

    <h2>Uma unica variavel</h2>

    <%= dados[0] %>

</body>

Modifique cadastro.ejs para iterar por todo o array

<body>

    <h1>Cadastro</h1>

    <h2>Uma unica variavel</h2>

    <%= dados[0] %>

    <h2>Iterando um array</h2>

    <br>

    <% dados.forEach((linha)=>{ %>

         <!-- posso escrever html aqui -->

        <b> <%= linha %>  </b> <br>

    <%})%>

</body>

1. Passando um objeto literal para uma view

Modifique novamente a função Cadastro de formController para enviar o objeto literal

    res.render('cadastro',pessoa)

e a view cadastro.ejs para mostrar um mostrar o nome da pessoa

<body>

    <h1>Cadastro</h1>

    <h2>Uma unica variavel</h2>

    <%= pessoa.nome %>

    <h2>Iterando um array</h2>

    <br>

    <% dados.forEach((linha)=>{ %>

         <!-- posso escrever html aqui -->

        <b> <%= linha %>  </b> <br>

    <%})%>

</body>

# Criando um model

É função do Model conversar com o banco e devolver os dados necessários

1. Crie a pasta .\models
2. Crie o arquivo exemploModel.js

let dados = ['dado1','dado2','dado3','dado4','dado5', 'dadosx']

exports.getDados = () => {

    console.log('exemploModel getDados()')

    return dados

}

exports.setDados = (dado) =>{

    dados.push(dado)

    return true

}

1. Altere a função cadastro de formController

exports.cadastro = (req,res) =>{

    //res.send('formController, homeview()');

    console.log('formController cadastro()')

    // dados = ['dado1','dado2','dado3','dado4']

    exemploModel.setDados('teste')

    dados = exemploModel.getDados()

    pessoa = {

        nome: 'josue micaroni',

        idade: '45'

    }

    res.render('cadastro',dados)

}

# Criando um banco

1. Crie o banco (Foi usado xampp durante este tutorial)

create database nodedb;

use nodedb;

CREATE TABLE teste (campo1 varchar(255));

insert into teste(campo1) values('dbdado1');

insert into teste(campo1) values('dbdado2');

insert into teste(campo1) values('dbdado3');

insert into teste(campo1) values('dbdado4');

1. Altere o exemploModel para criar a conexão com o banco e retornar dados de um select

const db = require('../models/banco');

// let dados = ['dado1','dado2','dado3','dado4','dado5', 'dadosx']

dados = []

exports.getDados = () => {

    console.log('exemploModel getDados()')

    db.getConnection() // abre a conexão com o banco retorna a conexão(uma promisse)

    .then(conn => {

       // usa a conexao para fazer uma query ( retorna como promisse)

      conn.query("select \* from teste")

        .then((rows) => {

          console.log(rows);

          dados = rows

          conn.end(); // fecha a conexão com o banco

        })

        .catch(err => {

          //trata os erros da query

          console.log(err);

          conn.end();

        })

    }).catch(err => {

      //trata os erros da conexão

    });

    return dados

}

exports.setDados = (dado) =>{

    dados.push(dado)

    return true

}

Cada linha do select é um objeto. O retorno é um array de objetos. É necessário alterar a view para mostrar corretamente os objetos

Altere a view cadastro.ejs ( linha passa a ser linha.campo1)

    <h2>Iterando um array</h2>

    <br>

    <% dados.forEach((linha)=>{ %>

         <!-- posso escrever html aqui -->

        <b> <%= linha.campo1 %>  </b> <br>

    <%})%>

# ACRESCENTANDO NOVOS CAMPOS

Criar uma nova tabela

USE nodedb;

CREATE OR REPLACE TABLE pessoa (

id MEDIUMINT NOT NULL AUTO\_INCREMENT,

nome varchar(100),

telefone varchar(14),

endereco varchar(255),

email varchar(255),

cpf varchar(12),

rg varchar(12),

PRIMARY KEY (id)

);

INSERT INTO pessoa ( nome, telefone,endereco, email, cpf, rg) VALUES

('José Cansado da lida', '19977889900', 'Rua das duas uma, 154, Campinas/sp', 'jose@email.com.br','22233344455','223344550'),

('Janete Claudete Limete', '19966777832', 'Rua dos desalmados, 689, Campinas/SP', 'janetelindinha@email.com.br','23515896574','698974598'),

('Joao Subiro Nakombi', '199587823596', 'Rua das avenidas, 125, Hortolandia/SP', 'subirojoao@email.com.br','23569865841','363598956'),

('Julia Cavaro Buraco', '19966582325', 'Rua das caras bravas, 1256, Sumaré/SP','julia@email.com.br','12569874584','235698458');

.\App.js

Acrescentar as linhas configuração do bootstrap

//configuração do bootstrap

app.use('/css', express.static(path.join(\_\_dirname, 'node\_modules/bootstrap/dist/css')))

app.use('/js', express.static(path.join(\_\_dirname, 'node\_modules/bootstrap/dist/js')))

app.use('/js', express.static(path.join(\_\_dirname, 'node\_modules/jquery/dist')))

\views\cadastro.ejs

Acrescentar a linha para inserir o css no código

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="./css/bootstrap.min.css">

    <title>Cadastro</title>

</head>

Acrescentar uma table simples do bootstrap

Obs.: Copie o código diretamente da página do bootstrap e altere para os novos campos

<https://getbootstrap.com/docs/5.3/content/tables/>

    <table class="table">

        <thead>

          <tr class="table-dark" >

            <th scope="col">#</th>

            <th scope="col">Nome</th>

            <th scope="col">Telefone</th>

            <th scope="col">Endereco</th>

            <th scope="col">Email</th>

            <th scope="col">CPF</th>

            <th scope="col">RG</th>

          </tr>

        </thead>

        <tbody>

          <tr>

            <th scope="row ">1</th>

            <td>Mark</td>

            <td>Otto</td>

            <td>@mdo</td>

          </tr>

      </table>

</body>

Alterar cadastro.ejs para acrescentar o forEach para os novos campos e os novos botões de CRUD

<body>

    <h1>Cadastro Pessoa</h1>

    <button type="button" class="btn btn-primary">Cadastrar</button>

    <table class="table">

        <thead>

          <tr class="table-dark" >

            <th scope="col">#</th>

            <th scope="col">Nome</th>

            <th scope="col">Telefone</th>

            <th scope="col">Endereco</th>

            <th scope="col">Email</th>

            <th scope="col">CPF</th>

            <th scope="col">RG</th>

            <th scope="col">Ação</th>

          </tr>

        </thead>

        <tbody>

            <% dados.forEach((linha)=>{ %>

                <!-- posso escrever html aqui -->

                <tr>

                    <th scope="row "><%= linha.id %></th>

                    <td><%= linha.nome %></td>

                    <td><%= linha.telefone %></td>

                    <td><%= linha.endereco %></td>

                    <td><%= linha.email %></td>

                    <td><%= linha.cpf %></td>

                    <td><%= linha.rg %></td>

                    <td>

                        <button type="button" class="btn btn-warning">Modificar</button>

                        <button type="button" class="btn btn-danger">Deletar</button>

                    </td>

                </tr>

           <%})%>

      </table>

</body>