Modulo 07

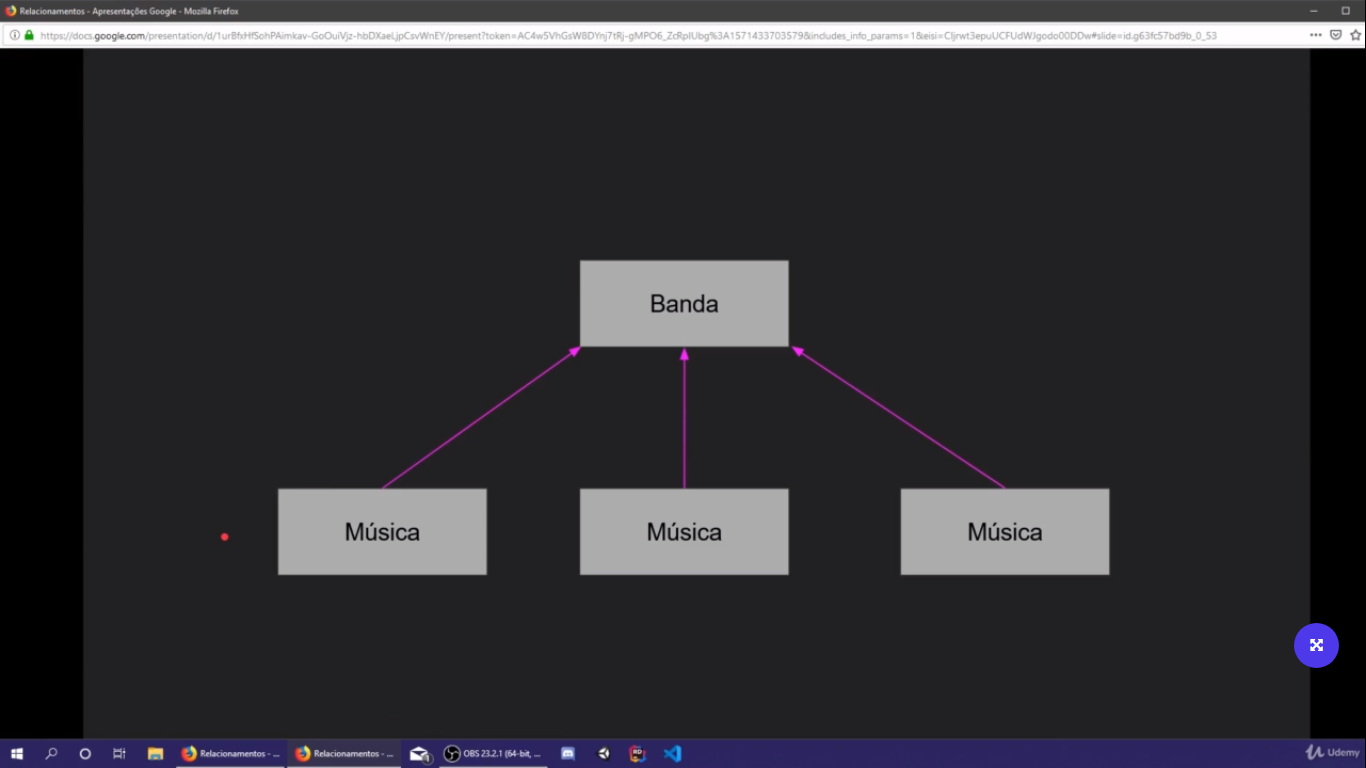
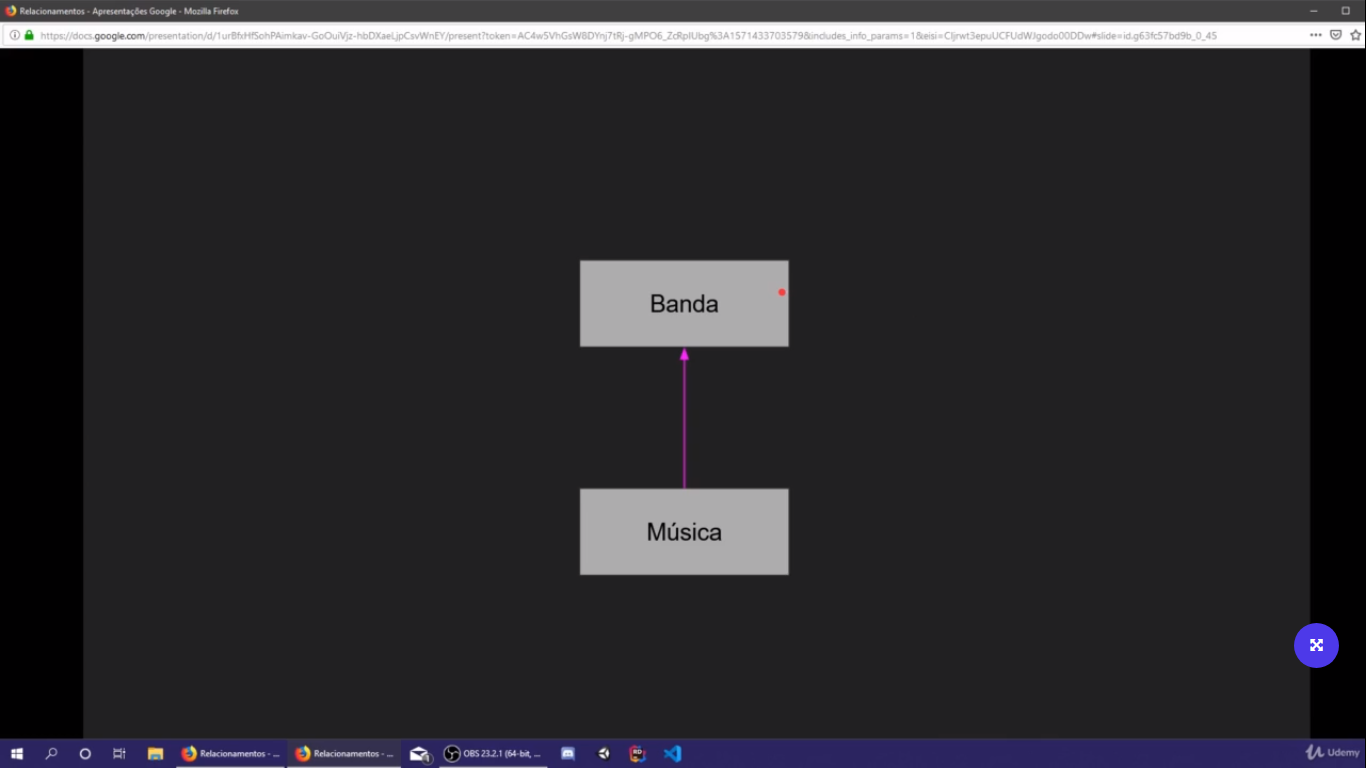
Projeto 02 Criando um blog com painel administrativo – CRUD completo

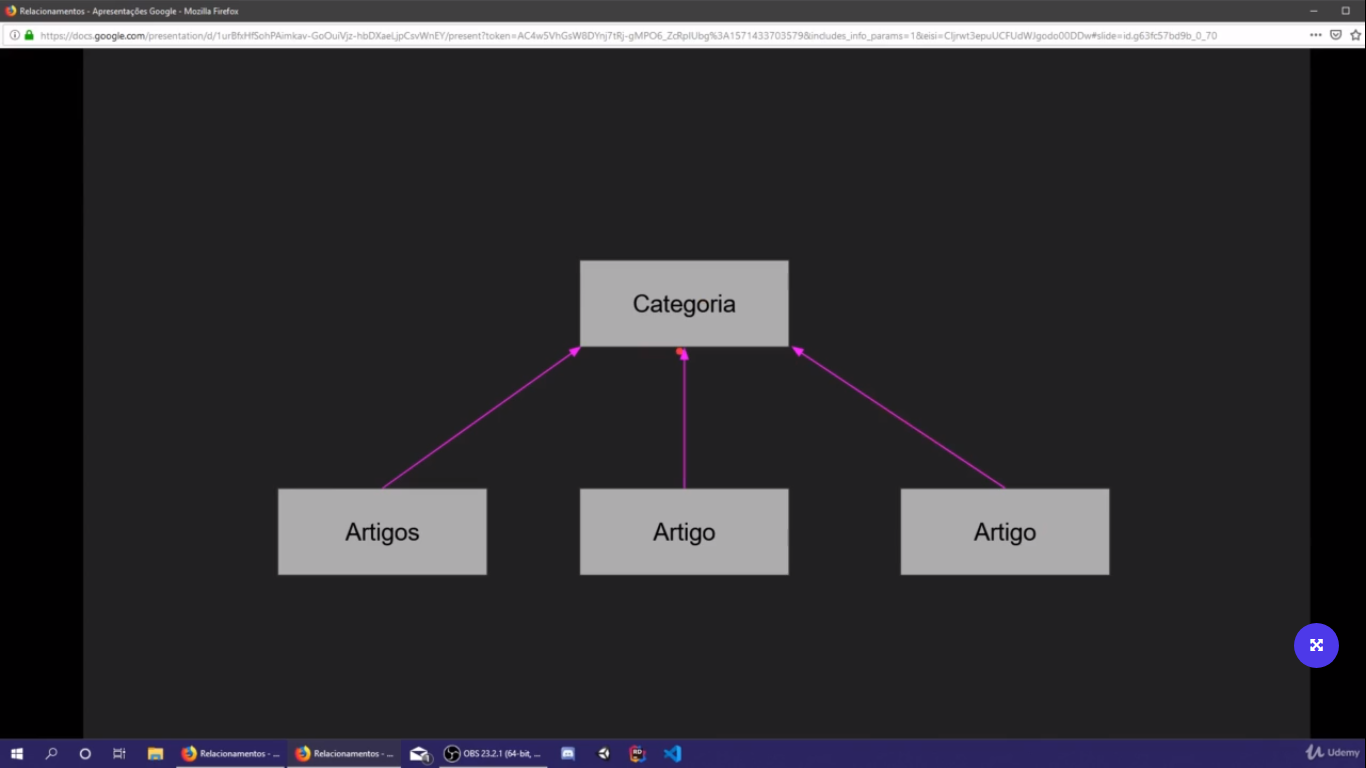
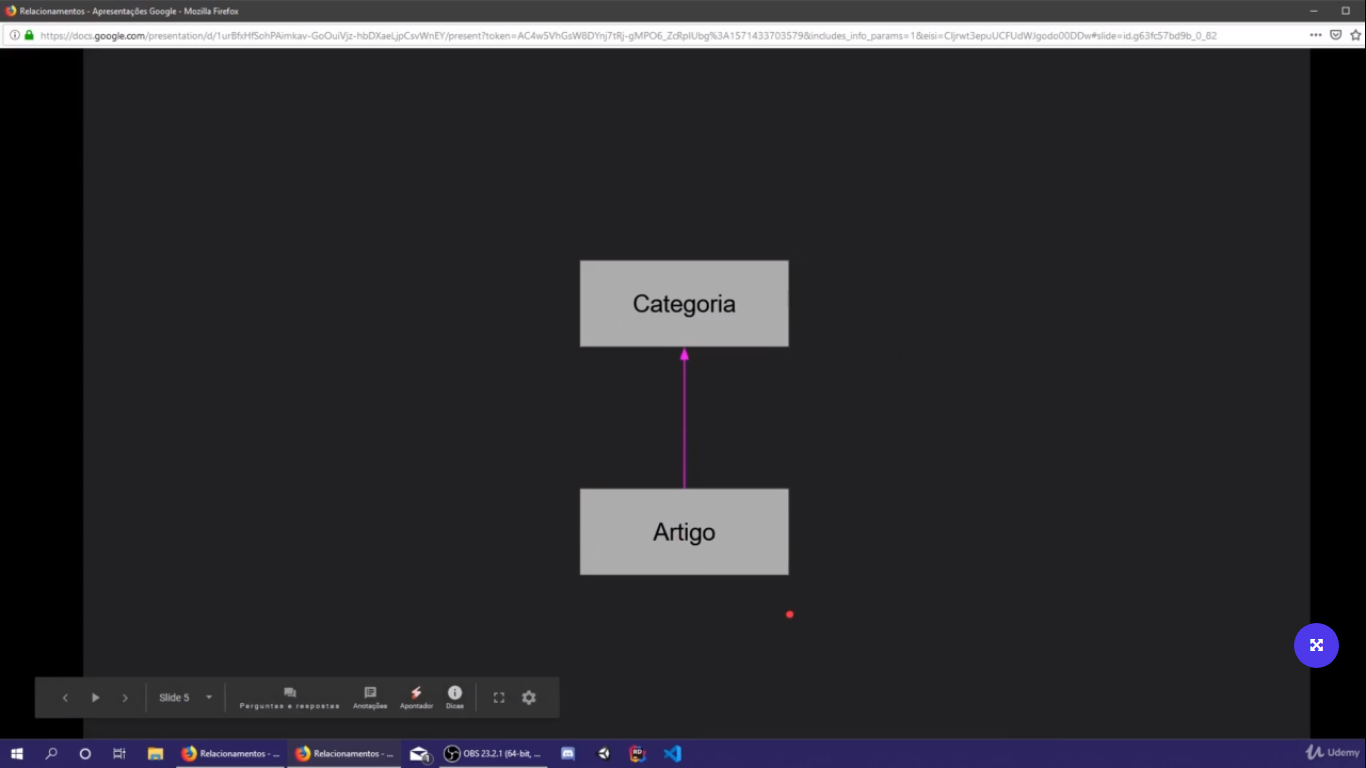
* Aula 01 Introdução

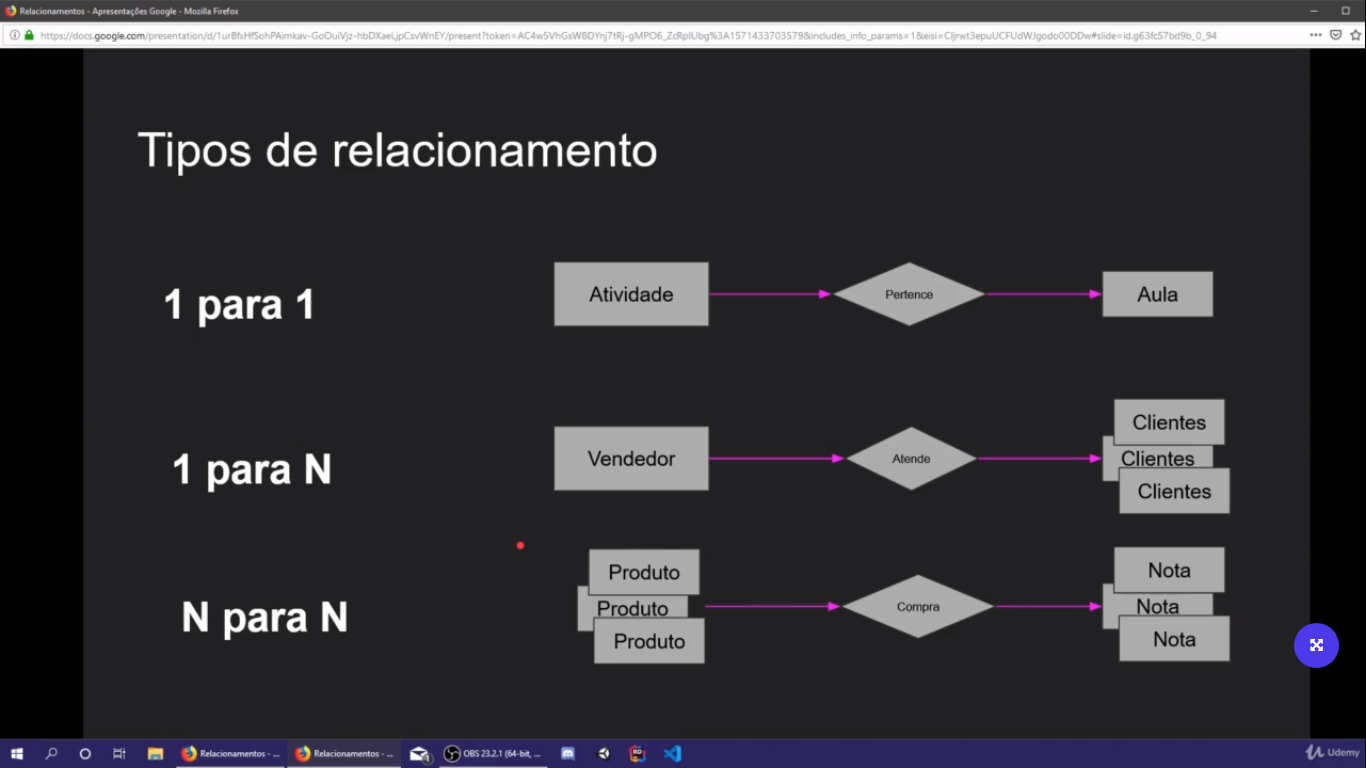
TinyMCE – Editor de texto mais utilizado para formatação de texto do estilo de blogs

* Aula 02 Relacionamento

Associação entre duas ou mais tabelas, ligar uma tabela conversar com uma outra







* Aula 03 Base do projeto

Ultilazaremos as bibliotecas, express, body-parser, ejs, sequelize e mysql2

Vamos configurar o projeto com bootstrap

Iremos criar o projeto na pasta guiaPress

Vamos começar!!!!

npm init -y

npm install --save express

npm install --save sequelize

npm install --save mysql2

npm install --save body-parser

npm install --save ejs

crie o arquivo index.js, e faça as configurações

index.js

const express = require('express')

const app = express()

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

    res.send('Bem vindo ao meu site')

})

app.listen(8080, () => {

    console.log('O Servidor está rodando na porta 8080');

})

Configure as:

- views engine(estrutura ‘html’ do projeto)

- body-parser(trabalha com formularios)

- Static(onde ficaram os arquivos estaticos)

- Criar seu banco de dados do projeto e suas conexões

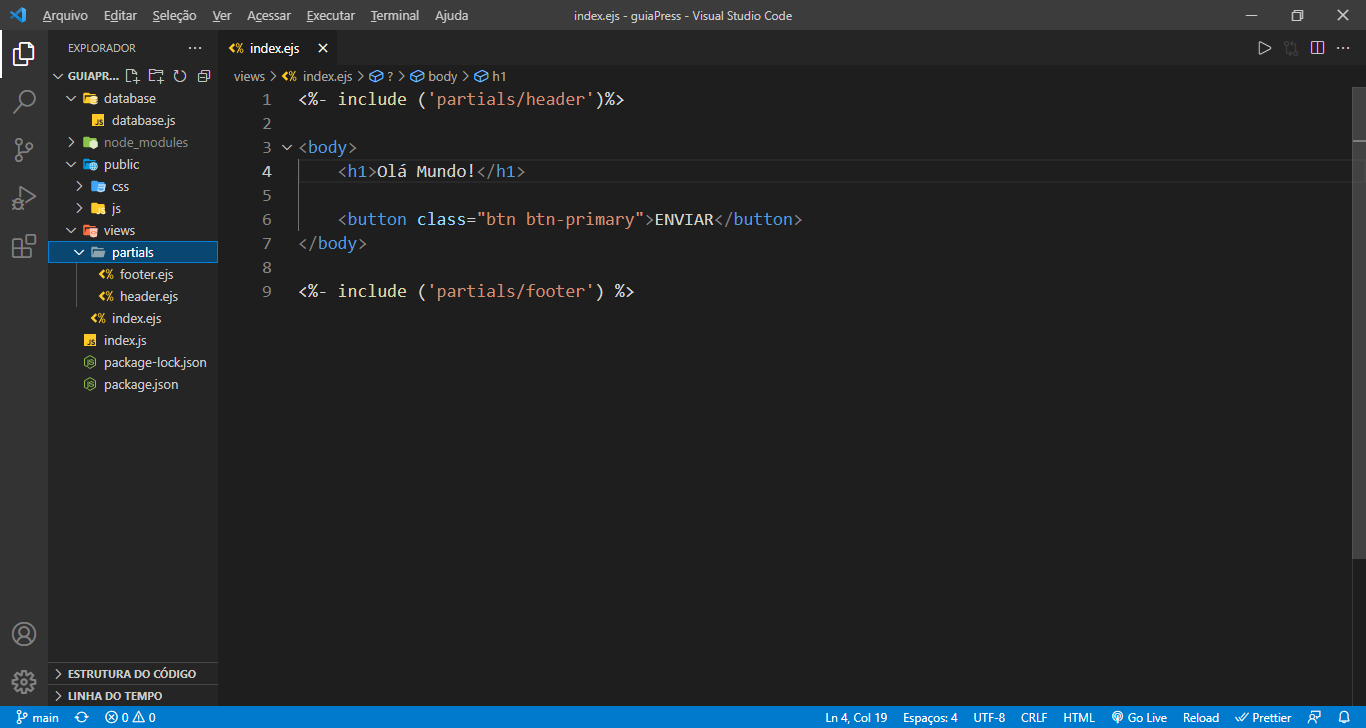
Feito essas configurações iremos instalar o bootstrap no projeto

Como estamos usando o ejs como o template engine, criamos arquivos parciais para o header e footer de nosso projeto

Na partials de header iremos fazer o link para nosso arquivo bootstrap.min.css, que baixamos e colocamos na pasta public/css

Já na partials footer, iremos colar os scripts do Jquery e Pooper que pegamos no site do bootstrap e por final lincar o script bootstrap.min.js que está na pasta public/js

Nosso projeto terá essa cara ao final da configurações iniciais:



- Banco de dados

Pasta database

- Arquivos publico/estáticos

Pasta public

- Visualização de paginas

Pasta views

-- dentro da pasta views temos nossas partials(header e footer)

- Arquivo principal do projeto

index.js

database.js

const Sequelize = require('sequelize')

const connection = new Sequelize('guiapress','root','123456',{

    host: 'localhost',

    dialect: 'mysql'

})

module.exports = connection

header.ejs

<!DOCTYPE html>

<html lang="pt-BR">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

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

    <title>GuiaPress</title>

    <!-- Importando o Bootstrap -->

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

</head>

footer.ejs

<!-- Carregando Jquery -->

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

<!-- Carregando Popper -->

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js" integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="anonymous"></script>

<!-- Carregando Bootstrap -->

<script src="/js/bootstrap.min.js"></script>

</html>

index.js

const express = require('express')

const app = express()

const bodyParser = require('body-parser')

const connection = require('./database/database')

// Viem Engine

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

// Static

app.use(express.static('public'))

// Body-parser

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

app.use(bodyParser.json())

// Database

connection.authenticate()

.then(() => {

    console.log('Conexão feita com sucesso');

})

.catch( err => {

    console.log('ERRO ao conectar ao BANCO DE DADOS '+ err);

})

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

    res.render('index')

})

app.listen(8080, () => {

    console.log('O Servidor está rodando na porta 8080');

})

* Aula 04 Arquitetura e organização de aplicação Node.js

O projeto e divido por dorminio, vamos usar um exemplo de um blog que tem dois domínios, o domínio de artigos e o domínio de categorias, são duas entidades diferentes, duas pastas diferentes

Controler -> nome do arquivo dado, onde ficara toda a logica da nossa aplicação, onde iram ficar nossas rotas

Todas as funcionalidades relacionadas a categoria do nosso projeto serão salvas dentro de scripts dentro dessa pasta que iremos criar com o nome de catogories

Iremos criar uma pasta chamada de articles, onde ficaram as logicas dos artigos

Agora para definir a rotas em um arquivo diferente ao arquivo principal iremos usar um recurso do express chamado express routers, que permite que você crie rotas sem a necessidade de trabalhar com a variável app

Na pasta categories iremos criar um arquivo CategoriesController.js

CategoriesController.js

const express = require('express')

const router = express.Router()

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

    res.send('Rota de categorias')

})

router.get('/admin/categories/new', (req, res) => {

    res.send('Rota para criar uma nova categoria!')

})

module.exports = router

e iremos linkar ao nosso arquivo principal

index.js

const CategoriesController =

require('./categories/CategoriesController')

app.use( '/' , CategoriesController)

agora toda vez que acessamos nosso projeto que tenha o caminho indicado ele ira buscar na nossa nova variável que recebe as routes de outro arquivos

prefixo, e algo que vai somar a essa rota passada.

Se nesse prefixo tivesse ‘/antes’, as rotas no navegador terias que se inicidadas com /antes e o caminho de uma rota

<http://localhost:8080/antes/categories>

* Aula 05 Model de categoria

Iremos criar nosso model de categoria dentro da pasta que corresponde a ela, no caso na pasta categories, cada coisa ficando no seu lugar

Boa pratica, sempre que for definir um model, defina ele com a primeira letra maiúscula e sempre no singular

Category.js

const Sequelize = require('sequelize')

const connection = require('../database/database')

const Category = connection.define('categories', {

    title: {

        type: Sequelize.STRING,

        allowNull: false

    },

    slug: {

        type: Sequelize.STRING,

        allowNull: false

    }

})

module.exports = Category

slug => nome da aplicação VIRA nome-da-aplicacao

* Aula 06 Model de artigo

Iremos criar nosso model de artigo dentro da pasta articles

Article.js

const Sequelize = require('sequelize')

const connection = require('../database/database')

const Article = connection.define('articles', {

    title:{

        type: Sequelize.STRING,

        allowNull: false

    },

    slug: {

        type: Sequelize.STRING,

        allowNull: false

    },

    body: {

        type: Sequelize.TEXT,

        allowNull: false

    }

})

module.exports = Article

* Aula 07 Relacionamento na nossa aplicação

PERTENCE

ARTIGO

CATEGORIA

Todo ARTIGO PERTENCE a uma CATEGORIA 1 para 1

Uma CATEGORIA pode ter vários ARTIGOS 1 para N

* Aula 08 Definindo relacionamento no Sequelize

Escolha um model para fazer o reacionamento(article ou category)

Primeira coisa para fazer um relacionamento e importar com qual model ela que ser relacionar

Article.js

const Category = require('../categories/Category')

agora para dizer que o Article pertence a uma Category usamos o código belongTo(), essa e forma de representar o relacionamento 1 para 1

agora iremos usar o relacionamento 1 para muitos usando o hasMany(), forma de representar 1 para muitos

Article.belongsTo(Category) // um Articles pertence á uma Category

Category.hasMany(Article)   // Uma Category tem muitos Articles

Você poderia usar apenas um que não faria diferença, funcionaria do mesmo jeito, dessa forma foi feito um relacionamento de mao dupla

Sempre que e feito um novo relacionamento e necessário atualizar seu banco de dados, para isso vamos em todos os model que fazer esse relacionamento e forçar a sua sincronização

Article.js

Article.sync({force:true})

Category.js

Category.sync({force:true})

Toda a vez que utilizado esse método de sync com o force, ele irá criar uma nova tabela em seu banco, por isso na primeira vez que essa tabela for criada, você devera apagar o código, se não ele ira tentar forca a criação de uma nova tabela

Mais antes disso e preciso importa esses models no nosso arquivo principal

index.js

// Importando os Models

const Article = require('./articles/Article')

const Category = require('./categories/Category')

agora sim remova o códigos de sync de seus models, se não a cada vez que você rodar sua aplicação ele ira tentar criar essas tabelas

Category.js

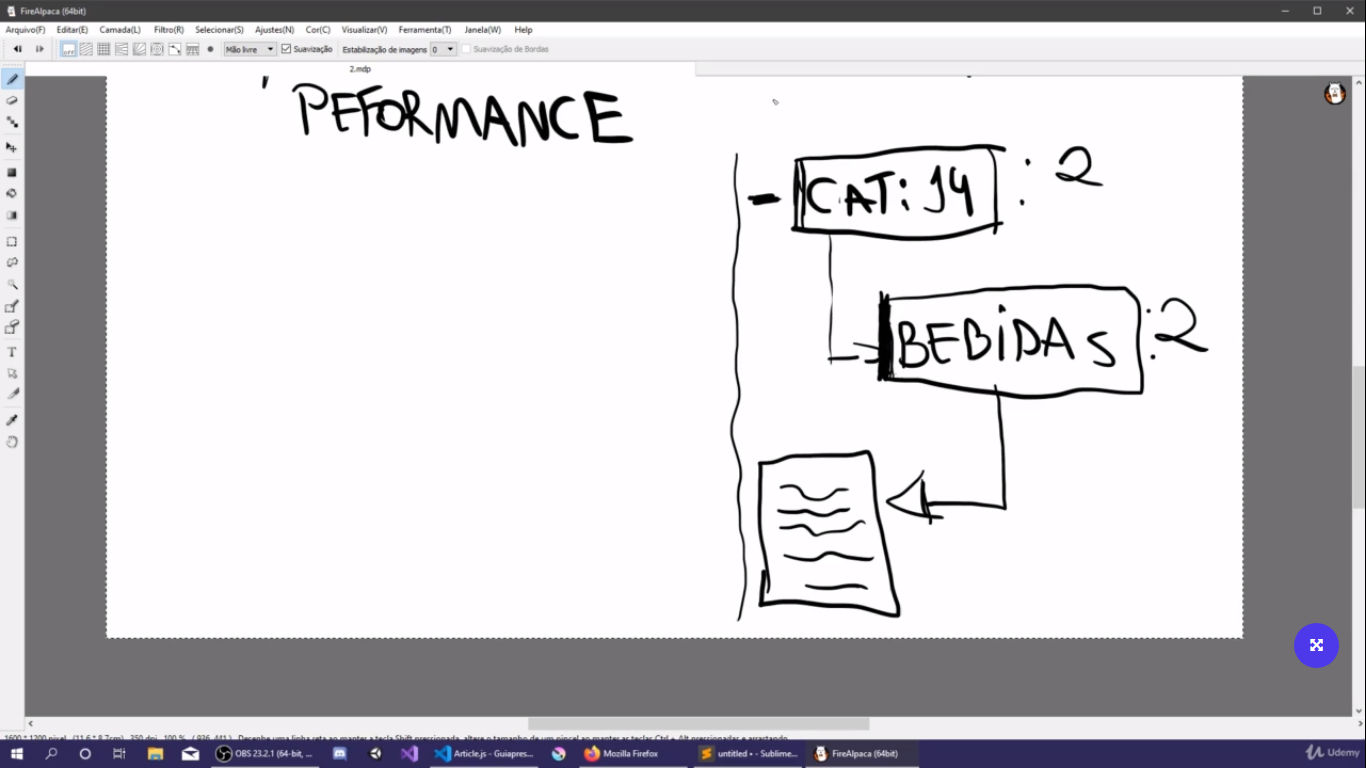
//Category.sync({force:true}) APAGAR

Article.js

//Article.sync({force:true}) APAGAR

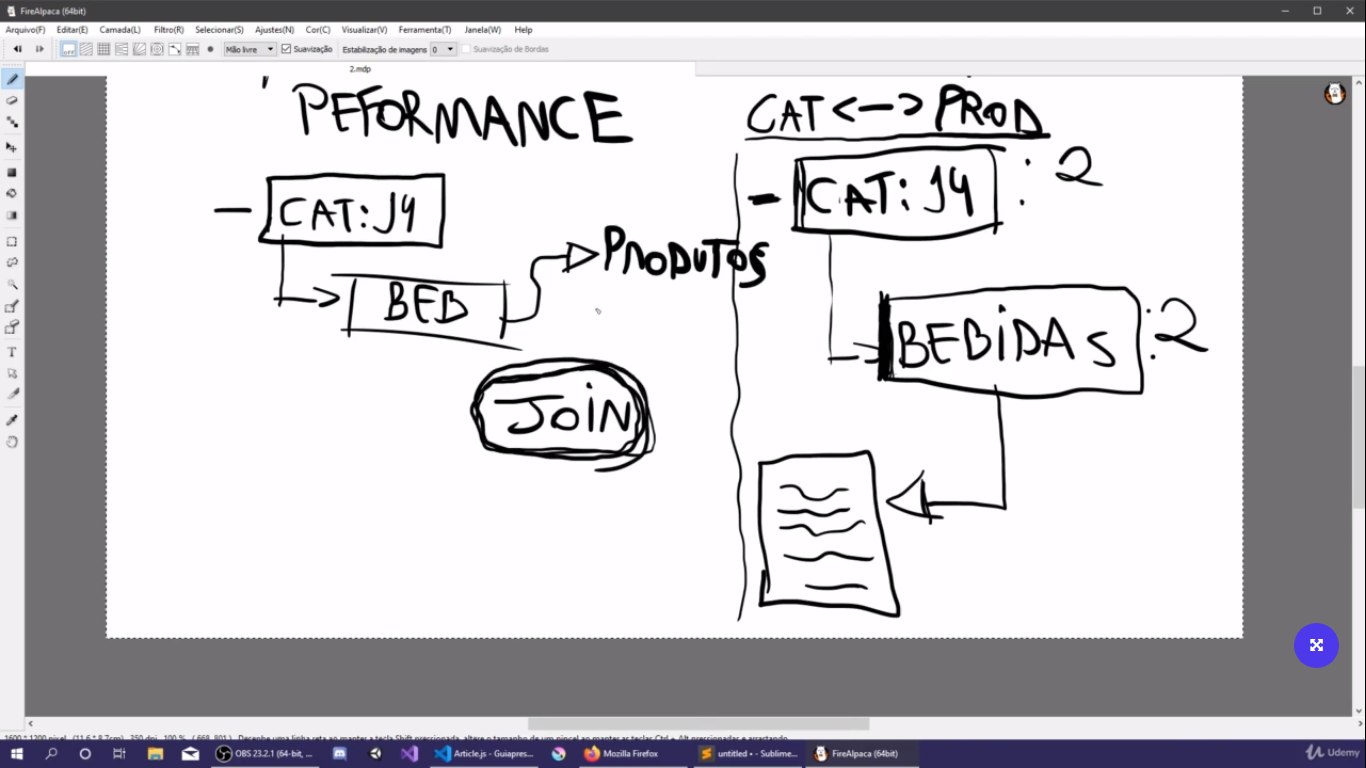
* Aula 09 qual a utilidade dos relacionamentos?

Para se ter uma melhor perfomace em sua aplicação



aqui você esta fazendo duas consultas em seu banco, o que poderia ser somente um,

isso em banco com muitos acessos pode ocacionar uma baixa performace

com o uso do JOIN conseguimos retornar a lista de produto de forma mais rápida, ganhando mais performace.

* Aula 10 Formulario de categoria

Criamos um caminho, que renderiza uma nova view

CategoriesController.js

router.get('/admin/categories/new', (req, res) => {

    res.render('admin/categories/new')

})

Criando essa nova view, dentro da pasta view -> admin -> categories -> new.ejs

new.ejs

<%- include ('../../partials/header')%>

<%- include ('../../partials/navbar') %>

<div class="container">

    <hr>

    <div class="card">

        <div class="card-header">

            <h2>Cadastro de Categorias</h2>

      </div>

        <div class="card-body">

            <form action="">

                <input class="form-control" type="text" name="title" placeholder="Defina o Título da categoria">

                <br>

                <button class="btn btn-success">Cadastrar</button>

            </form>

        </div>

    </div>

</div>

<%- include ('../../partials/footer') %>

Como estamos usando bootstrap e necessário presta atenção que nossas partials estão em outro diretório, e preciso colocar o caminho certo para ele

Nosso arquivo new.ejs está dentro da pasta

View -> admin -> categories -> new.ejs

E nossas partials estão mais acima desse diretório, por isso precisamos presta atenção com o caminho para chegar ate ele

<%- include ('../../partials/navbar') %>

* Aula 11 Cadastrando categorias e Slugify

Vamos instalar a bibliote slugify

npm install --save slugify

ela transforma uma string em slug, onde e possível ultiliar na rota de nosso navegador(sem espaço e caracteres especiais)

computação e informática PASSA A SER computacao-e-informatica

vamos carregar o slugify no arquivo que ira fazer essa alteração

new.ejs

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

   <input class="form-control" type="text" name="title" placeholder="Defina o Título da categoria">

  <br>

   <button class="btn btn-success">Cadastrar</button>

</form>

CategoriesController.ejs

const Category = require('./Category')

const slugify = require('slugify')

router.post('/categories/save', (req, res) => {

    let title = req.body.title

    if(title != undefined){

        Categoriy.create({

            title: title,

            slug: slugify(title)

        }).then(() => {

            res.redirect('/')

        })

    } else {

        res.redirect('/admin/categories/new')

    }

})

O formulário de new do método POST e enviado para o caminho /categories/save

No arquivo CategoriesController ele faz uma requisição do valor da tag que tem o name title e joga numa variável chamada com o mesmo nome, em sequida se esse valor for verdadeiro, ele faz um INSERT INTO na tabela Category passando o valor contido na variável title e no campo slug ele esta fazendo um slug no valor passado em title

* Aula 12 Tabelas do bootstrap

index.ejs

<%- include ('../../partials/header')%>

<%- include ('../../partials/navbar') %>

<div class="container">

    <hr>

    <h2>Categorias</h2>

    <hr>

    <table class="table table-bordered">

        <thead>

            <tr>

                <th>ID</th>

                <th>Título</th>

                <th>Slug</th>

                <th>Ações</th>

            </tr>

        </thead>

        <tbody>

           <tr>

               <th>1</th>

               <th>Dev Web</th>

               <th>dev-web</th>

               <th></th>

           </tr>

        </tbody>

    </table>

</div>

<%- include ('../../partials/footer') %>

Criamos um caminho em CategoryController que leva a essa pagina

router.get('/admin/categories', (req, res) => {

    res.render('admin/categories/index')

})

* Aula 13 Listagem de categorias

CategoriesController.ejs

router.get('/admin/categories', (req, res) => {

    Category.findAll().then(categories => {

        res.render('admin/categories/index', {categories: categories})

    })

})

Aqui estamos dando um SELECT \* FROM Category e jogando em uma variável chamado de categories, que em seguida e enviado ao render como atributo

index.ejs

<table class="table table-bordered">

        <thead>

            ...

        </thead>

        <tbody>

            <% categories.forEach(categories => { %>

                <tr>

                    <th><%- categories.id %> </th>

                    <th><%- categories.title %> </th>

                    <th><%- categories.slug %> </th>

                    <th></th>

                </tr>

            <% }) %>

        </tbody>

    </table>

Aqui estamos pegando os valores que contem na array que foi passado em categories e enviado cada valor em sua rodada

* Aula 14 Facilitando a navegação

index.ejs

 <tbody>

            <% categories.forEach(categories => { %>

                <tr>

                    <th><%- categories.id %> </th>

                    <th><%- categories.title %> </th>

                    <th><%- categories.slug %> </th>

                    <th>

                        <button class="btn btn-warning">Editar</button>

                        <button class="btn btn-danger">Excluir</button>

                    </th>

                </tr>

            <% }) %>

        </tbody>

* Aula 15 Deletando categorias

index.ejs

<% categories.forEach(category => { %>

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

    <input type="hidden" name="id" value="<%- category.id %>">

   <button class="btn btn-danger">Excluir</button>

</form>

<% }) %>

Aqui temos um botão dentro de um form que quando ativado, a pagina e enviada para o endereço ‘/categories/delete’, que contem um input que recebe o valor da array que foi percorrida, contem o nome id e como e do tipo hidden está escondida

CategoriesController.ejs

router.post('/categories/delete', (req, res) => {

    let identificador = req.body.id

Category.destroy( {where: {id: identificador } })

})

Ao enviar esse post os dados são direcionado a esse controller, que recebe na variável, identificador, o valor que continha no input com o name id

Em sequida temos o destroy(), que funciona basicamente com um

DELETE FROM <nome-tabela>

WHERE <condição>

Nesse caso seria:

DELETE FROM Category(que representa a tabela categories)

WHERE id = identificador;

Então ele esta apagando da tabela o campo que corresponde ao valor passado em identificador bate igual ao id

Ficando o código completo assim:

index.ejs

<% categories.forEach(category => { %>

<tr>

     <th><%- category.id %> </th>

     <th><%- category.title %> </th>

     <th><%- category.slug %> </th>

     <th>

       <button class="btn btn-warning" >Editar</button>

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

        <input type="hidden" name="id" value="<%-category.id%>">

          <button class="btn btn-danger">Excluir</button>

       </form>

     </th>

  </tr>

<% }) %>

CategoryController.ejs

router.post('/categories/delete', (req, res) => {

    let id = req.body.id

    if(id != undefined){

        if(!isNaN(id)){ // FOR um número

            Category.destroy({where:{id:id}})

.then(() => {

                res.redirect('/admin/categories')

            })

        } else { // Não for um número

            res.redirect('/admin/categories')

        }

    } else { // NULL

        res.redirect('/admin/categories')

    }

})

* Aula 16 Evitando acidentes

index.ejs

<form method="POST" action="/categories/delete"

onsubmit="confirmarDelecao(event)">

   <input type="hidden" name="id" value="<%- category.id %>">

   <button class="btn btn-danger">Excluir</button>

</form>

<script>

    function confirmarDelecao(event){

        event.preventDefault()

        console.log('o formulario nao foi enviado');

    }

</script>

Colocamos um evento do DOM, o onsubmit.

Esse onsubmit, e ativado no formulário cada vez que o formulário for submetido a alguma ação, então cada vez que o formulário e enviado ele executa uma função, a função confirmarDelecao que recebe um event como parâmetro

Nessa função cada vez que ele e ativada, a função normal do evento de onsubmit e cancelada, pois e passada o evento junto com o preventDefault()

Agora iremos usar um elemento do Javascript, onde aparece um popup duas opção ok e cancel, iremos usar o elemento confirm()

Nesse elemento e possível enviar um texto e caso ele seja escolhido OK retornara true caso cancel retornara false

let decision = confirm('Você quer deletar essa categoria')

agora iremos acessar nosso formulário por essa função

<form method="POST" action="/categories/delete"

onsubmit="confirmarDelecao(event, this)">

     <input type="hidden" name="id" value="<%- category.id %>">

     <button class="btn btn-danger">Excluir</button>

</form>

<script>

    function confirmarDelecao(event,form){

       event.preventDefault()

      console.log('o formulario nao foi enviado');

       let decision = confirm('Você quer deletar essa categoria')

      if(decision){ // Se o popup so verdadeiro

           form.submit()

      }

   }

</script>

Ao colocar o this, estamos passando o form como parâmetro, e caso o popup por respondido como verdadeiro ele ira envia o form.

Que nesse caso está recebendo o formulário completoe enviando através do submit()

* Aula 17 Formulario de edição

CategoryController.ejs

router.get('/admin/categories/edit/:id', (req, res) => {

    let id = req.params.id

    if(isNaN(id)){

        res.redirect('/admin/categories')

    }

    Category.findByPk(id).then( category => {

        if(category != undefined) {

            res.render('admin/categories/edit', {

category: category

})

        } else {

            res.redirect('/admin/categories')

        }

    }).catch( err => {

        res.redirect('/admin/categories')

    })

})

Criamos agora uma rota que recebe um parâmetro, esse parâmetro passado no endereço e pego e passado para uma variável chamada de id

Se essa variável id não for um numero, a pagina e redirecionada

Em sequida e feita uma busca na tabela que e representada pelo Category, ele faz uma busca pela chave primaria, que seja igual ao id passado Category.findByPk(id)

Se essa consulta retornar alguma coisa, ele ira criar uma arrow function e jogar o valor achado dentro de category

Caso esse valor achado seja verdadeiro, ele ira renderizar uma pagina, que recebera um atributo chamado de category que recebera o valor achado na arrow function que se chama category

edit.ejs

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

   <input name="title" value="<%= category.title %>">

   <br>

   <button class="btn btn-success">Atualizar</button>

</form>

Ao acessar essa rota no campo input, em value e definido o atributo que veio la da rota passado como category

* Aula 18 Linkando botões de edição

index.ejs

 <% categories.forEach(category => { %>

   <tr>

    <th>

      <a href="/admin/categories/edit/<%= category.id %> " class="btn btn-warning" >Editar</a>

      <form method="POST" action="/categories/delete" style="display: inline;" onsubmit="confirmarDelecao(event, this)">

        <input type="hidden"name="id" value="<%= category.id %>">

         <button class="btn btn-danger">Excluir</button>

       </form>

     </th>

   </tr>

<% }) %>

Alteramos o button para um a, na linha 4, que faz referencia ao nosso link de edição, e passsamos para esse link dinamicamente o valor contido em category.id, no seu laço de repetição do forEach

* Aula 19 Salvando edição

edit.ejs

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

<input class="form-control" type="text" name="title" placeholder="Defina o Título da categoria" value="<%= category.title %> ">

   <input type="hidden" name="id" value="<%= category.id %>">

   <br>

   <button class="btn btn-success">Atualizar</button>

</form>

Aqui temos um formulário que envia para o caminho /categories/update, contendo um input que se esconde na tela pois tem o tipo hidden com o nome de id, e um input com o nome title

CategoriesController.ejs

router.post('/categories/update', (req, res) => {

    let pegaId = req.body.id

    let novoTitle = req.body.title

    Category.update({title: novoTitle},{

        where: {

            id: pegaId

        }

    }).then(() => {

        res.redirect('/admin/categories')

    })

})

Ao enviar esse formulário, do código anterior, para essa rota do tipo post, estamos pegando dele os valores do tipo id e title e jogando em suas variaveis conrespondentes

Agora com esses valores vamos fazer as alterações em nossa tabela

Com o uso do update

Pegamos o nosso model de Category, e damos um update em seus valor, seria mesma coisa que caso digitarmos

UPDATE categories

SET title = novoTitle

WHERE id = pegaId

Porem colocamos o código do node

Category.update ({ title: novoTitle },{

        where: {

            id: pegaId

        }

    })

O que será feito

Onde será feito

Valor a ser inserido

Onde

Parâmetro igual há

Category.update({title: novoTitle, slug: slugify(novoTitle)},{

        where: {

            id: pegaId

        }

    })

Aqui estamos alterando dois valores, o title e o slug da tabela que recebe o slugify de novoTitle

* Aula 20 Formulario de artigo

Criamos um nova pasta de views/admin/articles, onde ficarao nossas paginas relacionadas ao tipo artigo

articles/new.ejs

<%- include ('../../partials/header')%>

<%- include ('../../partials/navbar') %>

<div class="container">

    <hr>

    <div class="card">

        <div class="card-header">

            <h2>Cadastro de Artigo</h2>

        </div>

        <div class="card-body">

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

                <input class="form-control" type="text" name="title" placeholder="Defina o Título do artigo">

                <br>

                <textarea class="form-control" placeholder="Escreva o artigo aqui!" name="body"></textarea>

                <br>

                <button class="btn btn-success">Cadastrar</button>

            </form>

        </div>

    </div>

</div>

<%- include ('../../partials/footer') %>

Temos um form com os input contendo o name title e body(onde ficará o corpo do artigo)

* Aula 21 Configurando o TinyMCE

<https://www.tiny.cloud/>

biblioteca para criação de editores de texto para html

baixe o tiny pelo link <https://www.tiny.cloud/get-tiny/self-hosted/>

baixe a opção Commnunity, extraia a pasta tinymce para a pasta public do nosso projeto

agora nossa pasta public terá as pasta css, js e tinymce

agora precisamos dizer qual textarea ganhará esse ‘SUPER PODER’ do tinymce

carregando o script ‘tinymce.min.js’ do tiny na views que se encotra essa textarea, e adicinonamos nessa texarea um id para podermos localizala no tiny, depois só configurar ele para pegar essa textarea pelo id e depois no script iniciar ele

articles/new.ejs

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

    <textarea id="article" name="body"></textarea>

</form>

<script src="/tinymce/tinymce.min.js"></script>

<script>

    tinymce.init({

        selector: '#article'

    })

</script>

E para melhorar nosso tiny podemos adicionar plugins nele

<script src="/tinymce/tinymce.min.js"></script>

<script>

    tinymce.init({

        selector: '#article',

        plugins:[

            'advlist autolink link image lists print preview hr searchreplace wordcount fullscreen insertdatetime media save table paste emoticons'

        ]

    })

</script>

Criamos uma list de plugins dentro do init, e separamos os plugins que queremos adicionar por espaço simples

<https://www.tiny.cloud/tinymce/features/>

você poderá achar mais plugins nesse link

* Aula 22 Traduzindo o TinyMCE

<https://www.tiny.cloud/get-tiny/language-packages/>

baixe o pacote de idioma que deseja nesse link, sera baixado um arquivo zip, dentro desse zip, teremos uma pasta langs e dentro dessa pasta teremos o arquivo que desejamos

copie esse arquivo e cole dentro da pasta de seu prejeto

\public\tinymce\langs

E em seguida adicionar nas configurações do tinymce o language seguido do código da linguage que deseja inserir, nosso caso o código do português do Brasil e pt\_BR

<script>

    tinymce.init({

        language: 'pt\_BR',

        selector: '#article',

        plugins:[

            'advlist autolink link image lists print preview hr searchreplace wordcount fullscreen insertdatetime media save table paste emoticons'

        ]

    })

</script>

Para arrastar e colar uma imagem direto coloque o código

tinymce.init({

...

paste\_data\_images: true

});

* Aula 23 Dropdown de categorias

ArticlesController.ejs

const Category = require('../categories/Category')

router.get('/admin/articles/new', (req, res) => {

    Category.findAll().then(categories => {

        res.render('admin/articles/new', {categories: categories})

    })

})

Estamos importando o model Category para pode fazer uma requisição nele ao acessar a rota ‘/admin/articles/new’, ele pega esses valores e passa como atributo na rota que está sendo redirecionada

Article/new.ejs

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

<label >Categoria</label>

   <select class="form-control" name="category">

     <% categories.forEach(category => { %>

      <option value="<%= category.id %> ">

<%= category.title %>

</option>

     <% }) %>

   </select>

   <br>

   <button class="btn btn-success">Cadastrar</button>

</form>

Aqui estamos pegando esse atributo que foi passado com o nome de category.

E feito um forEach para ler cada campo dessa variável passada, que ira criar um campo option para cada valor dentro dessa pesquisa, ao criar um optin seu value será o id e seu campo sera o seu title

* Aua 24 Salvando artigos

Articles/new.ejs

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

<input name="title" placeholder="Título do artigo">

   <textarea id="article" name="body" ...></textarea>

   <br>

   <label >Categoria</label>

   <select class="form-control" name="category">

    <% categories.forEach(category => { %>

       <option value="<%= category.id %>">

<%= category.title %>

</option>

       <% }) %>

    </select>

    <br>

   <button class="btn btn-success">Cadastrar</button>

</form>

ArticlesController.ejs

router.post('/articles/save', (req, res) => {

    let title = req.body.title         // titulo do artigo

    let body = req.body.body           // corpo do artigo

    let category = req.body.category   // id do artigo

    Article.create({

        title: title,

        slug: slugify(title),

        body: body,

        categoryId: category

// Chave estrangeira, temos essa chave porque fizemos

        // esse relacionamento Article.belongsTo(Category)

    }).then(() => {

        res.redirect('/admin/articles')

    })

})

Estamos criando uma tabela com os valores enviado para o model Article e quando salvo estamos sendo redirecionado para a rota /admin/articles

* Aula 25 Ajustando tempo no Sequelize

Ao criar uma tabela, o mysql, cria dois campo, que são:

createdAt – que mostra o horário que o campo foi criado

updateAt – Mostra o horário que o campo foi modificado

essa hora por padrão e usado o UTC(padrão universal), porem errado para o Brasil, para isso precisamos alterar essa timezone para a do Brasil

verifique em qual timezone irá ser o padrão do seu projeto

<https://www.timeanddate.com/>

em sequida no arquivo do nosso projeto que representa a criação do banco de dados iremos adicionar o código de timezone

database.js

const Sequelize = require('sequelize')

const connection = new Sequelize('guiapress','root','123456',{

    host: 'localhost',

    dialect: 'mysql',

    timezone: '-03:00'

})

module.exports = connection

* Aula 26 Pagina de artigos

Criamos uma pagina em view/articles/index

E lincamos essa pagina ao acessa o caminho /admin/articles

ArticlesController.ejs

router.get('/admin/articles', (req, res) => {

    res.render('admin/articles/index')

})

* Aula 27 Corrigindo bug da navbar

navbar.ejs

<nav class="navbar navbar-dark bg-primary navbar-expand-lg">

    <a class="navbar-brand" href="">

        GuiaPress

    </a>

    <ul class="navbar-nav mr-auto">

        <li class="nav-item">

            <a class="nav-link" href="/admin/categories">Categorias</a>

        </li>

        <li class="nav-item">

            <a class="nav-link" href="/admin/articles">Artigos</a>

        </li>

    </ul>

</nav>

Colocamos o código do bootstrap para alinhar os itens do navbar,

Usamos o codigo navbar-expand-lg

* Aula 28 Tabela de artigos

articles.ejs

<tbody>

   <% articles.forEach(article => { %>

    <tr>

         <td><%= article.id %> </td>

         <td><%= article.title %></td>

         <td><%= article.slug %></td>

         <td><%= article.categoryId %></td>

      <td>

      <button class="btn btn-warning">Editar</button>

     <button class="btn btn-danger">Apagar</button>

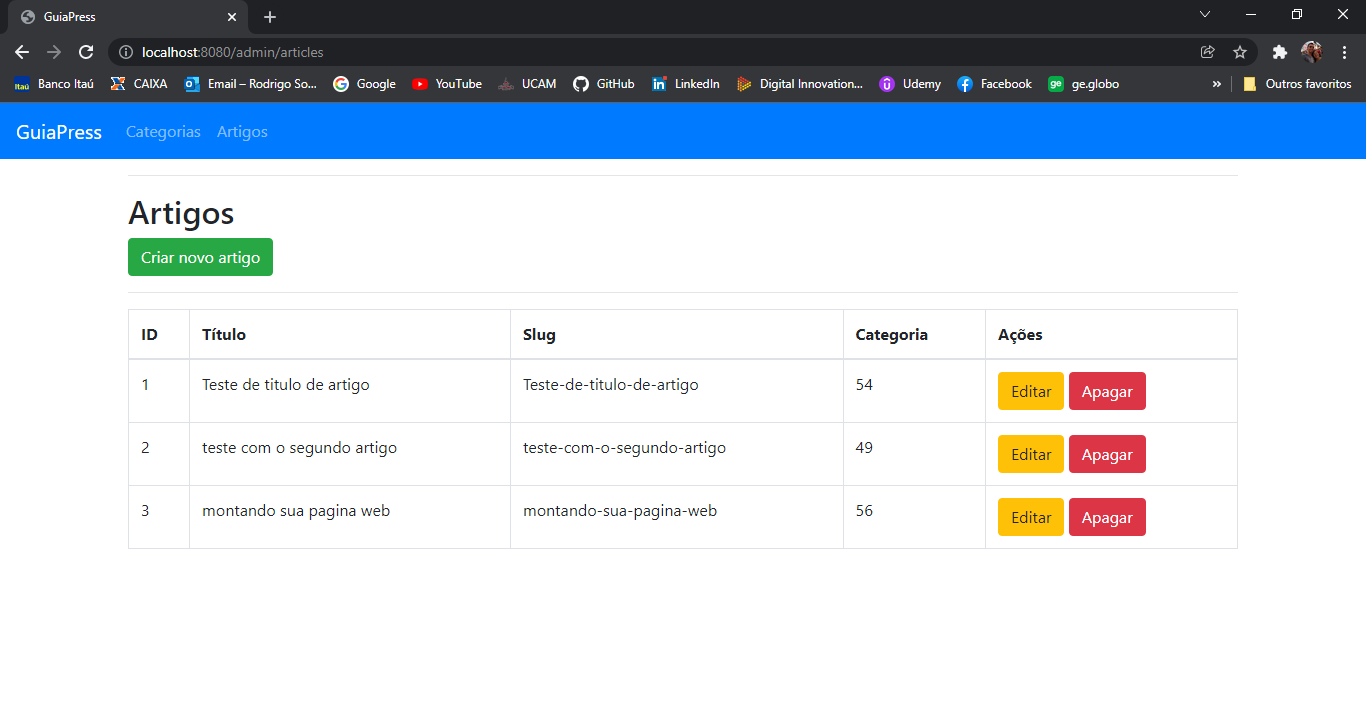
         </td>

      </tr>

   <% }) %>

</tbody>

No campo que pega o categoryId, sera motrado o valor do id no campo



* Aula 29 Join com sequelize

ArticlesController.ejs

router.get('/admin/articles', (req, res) => {

    Article.findAll({

        include:[{model:Category}]

    }).then(articles => {

        res.render('admin/articles/index', {articles: articles})

    })

})

Em nossa busca em todo o model de Article, estamos incluindo o model Category nessa busca

articles/index.ejs

<% articles.forEach(article => { %>

<tr>

<td><%= article.id %> </td>

<td><%= article.title %></td>

<td><%= article.slug %></td>

<td><%= article.category.title %></td>

<td>

<button class="btn btn-warning">Editar</button>

<button class="btn btn-danger">Apagar</button>

</td>

</tr>

<% }) %>

Nesse campo estamos recebendo a array article e nela estamos pegando a tabela category e pegando o valor de title

articles

PK id

categories

tem

FK categoryId

PK id

title

title

slug

body

slug

<td><%= article.category.title %></td>

Esse código corresponde ao código de join

SELECT articles.title

FROM articles

INNER JOIN categories

ON articles.categoryId = categories.id

Em outra palavras podemos dizer que estamos pedindo

para mostrar o title da tabela categories que fazem ligação com a tabela articles

* Aula 30 Deletando artigos

router.post('/articles/delete', (req, res) => {

    let id = req.body.id

    if(id != undefined){

        if(!isNaN(id)){  // FOR um número

            Article.destroy({

                where: {

                    id: id

                }

            }).then( () => {

                res.redirect('/admin/articles')

            })

        } else { // Não for um número

            res.redirect('/admin/articles')

        }

    } else {     // NULL

        res.redirect('/admin/articles')

    }})

<form method="POST" action="/articles/delete" style="display:

inline;" onsubmit="confirmarDelecao(event, this)">

<input type="hidden" name="id" value="<%= article.id %>">

   <button class="btn btn-danger">Excluir</button>

</form>

<%- include ('../../partials/footer') %>

<script>

    function confirmarDelecao(event,form){

        event.preventDefault() // cancela envio do form

        console.log('o formulario nao foi enviado');

        let decision = confirm('Você quer deletar este artigo?')

        if(decision){ // Caso verdadeiro

            form.submit()

        }

    }

</script>

Função que intercepta o envio do formulário

* Aula 31 Home page do blog e correção do bug da tabela

index.js

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

    Article.findAll().then(articles => {

        res.render('index', {articles: articles})

    })

})

Pegando todos os campos da tabela que estão no model de Article e jogando para o endereço index.ejs que está na pasta de view

view/index.ejs

<%- include ('partials/header')%>

<%- include ('partials/homenavbar') %>

<body>

    <div class="container">

        <% articles.forEach(article => { %>

            <div class="card">

                <div class="card-header">

                    <h2><%= article.title %> </h2>

                </div>

                <div class="card-body">

                    <a class="btn btn-success" href="#">

Ler Artigo

</a>

                </div>

            </div>

            <br>

        <% }) %>

    </div>

</body>

<%- include ('partials/footer') %>

Como foi passado o atributo articles para essa view, podemos acessala e recuperar os valores com o forEach(), e para cada volta no atributo articles criamos um novo campo em nossa div

* Aula 32 Pagina de leitura de artigo

Criamos uma rota que recebe um parâmetro obrigatório(iremos usar o slug para isso), criamos uma variável que recebe esse parâmetro com o name de slug, em seguida fazemos uma busca no model de Article que procura na tabela em um único campo o valor onde o campo slug seja igual ao slug passado, ao receber, esse valor e enviado como um atributo article para a rota article

e criamos um arquivo artile.ejs dentro da pasta view que recebe esse parâmetro de article mencionado

index.js

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

    let slug = req.params.slug

    Article.findOne({

        where: {

            slug: slug

        }

    }).then( article => {

        if (article != undefined) {

            res.render('article', {article: article})

        } else {

            res.redirect('/')

        }

    }).catch(err => {

        res.redirect('/')

    })

})

view/article.ejs

<%- include ('partials/header')%>

<%- include ('partials/homenavbar') %>

<body>

    <div class="container">

        <hr>

        <div class="card">

            <div class="card-header">

                <h2><%= article.title %> </h2>

            </div>

            <div class="card-body">

                <% article.body %>

            </div>

        </div>

    </div>

</body>

<%- include ('partials/footer') %>

* Aula 33 Linkando pagina de leitura

view/index.ejs

<a class="btn btn-success" href="/<%= article.slug %>">

Ler Artigo

</a>

Aqui colocamos um link dinâmico no href, cada link recebera o valor contigo no slug

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

    let slug = req.params.slug

esse link dinâmico ira ser enviado para essa parte do código

* Aula 34 Exibindo os conteúdos dinâmicos com EJS

Para resolver o problema do corpo do artigo que não estão formatado iremos usar essa solução

<div class="card-body">

    <%= article.body %>

</div>

Desse formato com estávamos ultiliando antes, iriamos receber uma variável do jeito que ela e escrita sem a formatação do html

Ex:

<h1><span style="text-decoration: underline;">com montar um site</span></h1> <p>aprendendo html voce ja poder&aacute; montar sua primeira pagina html😃</p> <p>&nbsp;</p>

<div class="card-body">

    <%- article.body %>

</div>

Exiba essa variável formatando ela com HTML, quando usamos o

<%**-** %>

# **com montar um site**

aprendendo html voce ja poderá montar sua primeira pagina html😃

logo com o uso do EJS podemos dizer que queremos usar a formatação HTML nessa variável em nosso projeto

* Aula 35 Menu dinâmico de categorias

index.js

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

    Article.findAll( {

        order: [

            ['id','DESC']

        ]

    }).then(articles => {

        Category.findAll().then( categories => {

            res.render('index', {articles: articles, categories: categories})

        })

    })

})

homenavbar.ejs

<ul class="navbar-nav mr-auto">

        <li class="nav-item">

            <a class="nav-link" href="/">Home </a>

        </li>

        <% categories.forEach(category => { %>

            <li class="nav-item">

                <a class="nav-link" href="/">

<%= category.title %>

</a>

            </li>

        <% }) %>

    </ul>

* Aula 36 Filtrando artigos por categoria

index.js

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

    let slug = req.params.slug

    Category.findOne({

        where: {

            slug: slug

        },

        include: [{model: Article}] // JOIN com model de Article

    }).then(category => {

        if(category != undefined){

            Category.findAll().then(categories => {

                res.render('index', {articles: category.articles, categories: categories})

            })

        } else {

            res.redirect('/')

        }

    }).catch(err => {

        res.redirect('/')

    })

})

homenavbar.ejs

<ul class="navbar-nav mr-auto">

        <li class="nav-item">

            <a class="nav-link" href="/">Home </a>

        </li>

        <% categories.forEach(category => { %>

            <li class="nav-item">

                <a class="nav-link" href="/category/<%= category.slug %> "><%= category.title %> </a>

            </li>

        <% }) %>

    </ul>

* Aula 37 Desafio 01

Tente desenvolver um sistema de edição de artigos

* Aula 38 Pagina de edição de arquivos

articlesController.ejs

router.get('/admin/articles/edit/:id', (req, res) => {

    let id = req.params.id

    Article.findByPk(id).then(article => {

        if(article != undefined) {

            Category.findAll().then(categories => {

                res.render('admin/articles/edit', {categories: categories, article: article})

                //res.send('Teste')

            })

        } else {

            res.redirect('/')

        }

    }).catch(err => {

        res.redirect('/')

    })

})

* Aula 39 TinyMCE Editavel

<input class="form-control" type="text" name="title"

placeholder="<%= article.title %>">

<br>

<textarea id="article" class="form-control" placeholder="Escreva o artigo aqui!" name="body" rows="20">

<%= article.body %>

</textarea>

Definindo o conteúdo dos input de edição de árticos

* Aula 40 Select dinâmico de edição

Articles/edit.ejs

<select class="form-control" name="category">

<% categories.forEach(category => { %>

<% if(category.id == article.categoryId){  %>

         <option value="<%= category.id %> " selected>

<%= category.title %>

</option>

       <% } else { %>

             <option value="<%= category.id %> ">

<%= category.title %>

</option>

       <% } %>

   <% }) %>

</select>

Fazendo uma seleção do option de um select

* Aula 41 Persistindo alterações

articles/edit.ejs

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

   <input type="hidden" value="<%= article.id %>" name="id">

   <input class="form-control" type="text" name="title" value="<%= article.title %>">

   <br>

  <textarea id="article" class="form-control" placeholder="Escreva o artigo aqui!" name="body" rows="20"><%= article.body %>  </textarea>

   <br>

   <label >Categoria</label>

   <select class="form-control" name="category">

     <% categories.forEach(category => { %>

       <% if(category.id == article.categoryId){  %>

         <option value="<%= category.id %> " selected><%= category.title %> </option>

       <% } else { %>

           <option value="<%= category.id %> ">

<%= category.title %> </option>

       <% } %>

     <% }) %>

   </select>

   <br>

   <button class="btn btn-success">Atualizar</button>

</form>

Aqui temos um formulário que e enviado para o link articles/update, contendo os name: id, title, body, category

Na tag de select temos uma condição que compara para ver se o id e igual de article e de category, sendo igual ele coloca como selecionado ao entrar na pagina de edição

Todos os valores dos input e textarea são definido pelo uso do EJS que pegam as variáveis que são passadas e inseridas em seus campos ou internos ou de value

articlesController.ejs

router.post('/articles/update', (req, res) => {

    let id = req.body.id

    let title = req.body.title

    let body = req.body.body

    let category = req.body.category

    Article.update({ title: title, body: body, categoryId: category, slug: slugify(title)},{

        where: {

            id: id

        }

    }).then(() => {

        res.redirect('/admin/articles')

    }).catch(err => {

        res.redirect('/')

    })

})

Ao enviar esse form acima caímos nessa rota, que pega esses valores com os respectivos name e coloca em variáveis cada uma

Depois e feito um update no model de article, enviando os valores para a alteração onde o campo id da tabela seja igual ao campo id passado pelo form anterior

* Aula 42 Logica da paginação

Articles.findAndCountAll()

Ele vai pesquisar todos os elemento do banco de dados e além disso ele vai retornar uma quantidade de quantos elementos existem nessa tabela

Ex: se em uma tabela temos 200 artigos, ele ira retornar todos os artigos dessa tabela e a quantidade de artigos que no caso e 200

articlesController.ejs

router.get('/articles/page/:num', (req, res) => {

    let num = req.params.num

    Article.findAndCountAll().then(articles => {

        res.json(articles)

    })

})

Estamos procurando tudo que tem no model de Article e sua contagem de total e enviando para a variável articles

A resposta a essa pagina e um arquivo JSON que recebe essa variável articles contendo todas as linhas da tabela e sua contagem total

articlesController.ejs

router.get('/articles/page/:num', (req, res) => {

    let num = req.params.num

    Article.findAndCountAll({

        limit: 4

    }).then(articles => {

        res.json(articles)

    })

})

Aqui temos um modelo de paginação, onde temos um limite de 4 linhas que será mostrada em nossa busca, graça ao parâmetro limit

articlesController.ejs

router.get('/articles/page/:num', (req, res) => {

    let num = req.params.num

    Article.findAndCountAll({

        limit: 4,

        offset: 5

    }).then(articles => {

        res.json(articles)

    })

})

Aqui temos o parâmetro offset, será mostrado 4 linhas da tabela apartir da 5 linha da tabela

router.get('/articles/page/:num', (req, res) => {

    let page = req.params.num

    if(isNaN(page) || page == 1) {

        offset = 0

    } else {

        offset = parseInt(page) \* 4

    }

    Article.findAndCountAll({

        limit: 4,

        offset: offset

    }).then(articles => {

        res.json(articles)

    })

})

Pegando o valor passado no endereço e jogando na variável page

Caso esse page não for um numero ou seja 1 o offset sera 0(começar a ler do numero 0 a tabela), caso seja outro valor, esse valor passado será multiplicado por 4(quantos itens queremos ver por pagina) e então em nossa busca no model Article enviaremos o offset sendo esse valor que foi multiplicado, a tabela comecara por esse valor multiplicado

router.get('/articles/page/:num', (req, res) => {

    let page = req.params.num

    if(isNaN(page) || page == 1) {

        offset = 0

    } else {

        offset = parseInt(page) \* 4

    }

    Article.findAndCountAll({

        limit: 4,

        offset: offset

    }).then(articles => {

        let next

        if(offset + 4 >= articles.count){

            next = false

        } else {

            next = true

        }

        let result = {

            next: next,

            articles: articles

        }

        res.json(result)

    })

})

Nesse campo temos um controle que mostra se há ou não uma próxima pagina, ele pega o offset e soma 4 caso seja maior que a quantidade de articles ele envia para a variável next false, caso esse valor seja menor a variável next recebe true

Depois cria um array com os campos next que recebe a variável next e o campo article que recebe a busca de todos os campos e total do model de articles.Em seguida a resposta dessa pagina e uma arquivo JSON dessa variável contendo esses objetos

* Aula 43 Pagina

articles/page.ejs

<%- include ('../../partials/header')%>

<%- include ('../../partials/homenavbar') %>

<body>

  <div class="container">

     <% result.articles.rows.forEach(article => { %>

      <div class="card">

          <div class="card-header">

             <h2><%= article.title %> </h2>

          </div>

       <div class="card-body">

        <a class="btn btn-success" href="/<%= article.slug %>">

Ler Artigo

</a>

          </div>

       </div>

    <% }) %>

  </div>

</body>

<%- include ('../../partials/footer') %>

Nova pagina de que recebe 3 partiais,header, navbar e footer

<% result.articles.rows.forEach(article => { %>

       ... ...

<% }) %>

Nesse caso estamos pegando o atributo result, nesse atributo pegando o campo articles e como ele foi feito no

Article.findAndCountAll

Ele retorna dois campo o count e o rows

Count -> a quantidade de elemento que tem

Rows -> as listas de artigos

ArticlesController.ejs

Article.findAndCountAll().then(articles => {

        let next

        let result = {

            next: next,

            articles: articles

        }

        Category.findAll().then( categories => {

            res.render('admin/articles/page',

{ result: result , categories: categories })

        })

})

module.exports = router

ArticlesController.ejs

Article.findAndCountAll().then( articles => {

        let next

        let result = {

            next: next,

            articles: articles

        }

        Category.findAll().then( categories => {

            res.render('admin/articles/page',

{ result: result , categories: categories })

        })

})

module.exports = router

ArticlesController.ejs

router.get('/articles/page/:num', (req, res) => {

    let page = req.params.num

    if(isNaN(page) || page == 0) {

        offset = 0

    } else {

        offset = parseInt(page) \* 4

    }

    Article.findAndCountAll({

        // quando se trabalha com o findAndCountAll

        // ele retorna 2 coisa

        // count -> a quantidade de elemento que tem

        // rows  -> as listas de artigos

        limit: 4,

        offset: offset

    }).then(articles => {

        let next

        if(offset + 4 >= articles.count){

            next = false

        } else {

            next = true

        }

        let result = {

            next: next,

            articles: articles

        }

        Category.findAll().then(categories => {

            res.render('admin/articles/page', {result: result, categories: categories})

        })

    })

})

Aqui nessa rota de /articles/page/:num

1 - Pegamos o parâmetro num e jogamos numa variável

2 – definimos o valor do offset de acordo com o valor de page

3 – fazemos um pesquisa de findCountAll no model Article, passando para ele um limite de 4 por pesquisa e um ponto de inicio

4 – o valor dessa pesquisa e jogado em articles, e contem um valor de count e um valor de rows

5 – iniciamos uma variável next

6 – definimos o valor dessa variável next de acordo com uma comparação entre offset e articles.count

7 – iniciamos uma array result, que recebe dois campos, next e articles, que reebem seus respetivos valores

8 – fazemos uma nova pesquisa em Category com o findAll

9 – ao finalizar essa pesquisa jogamos seu retorno no campo categories

10 – renderizamos uma pagina na pasta view e passamos para ela dois atributos o result que recebe a array result e o atribuo categories que recebe os valores da pesquisa feita e encaminhada para categories

Esses 10 passos acontecem ao acessar esse caminho de rota indicado acima

* Aula 44 Limiando visualização na Home page

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

    Article.findAll( {

        order: [

            ['id','DESC']

        ],

        limit: 4

    }).then(articles => {

        Category.findAll().then( categories => {

            res.render('index', {articles: articles, categories: categories})

        })

    })

})

Caminho da rota principal que ao fazer a pesquisa, orderna pelo id de forma descressente e limit a pesquisa a somente 4 valores

index.ejs

<div class="row">

<div class="col col-12">

    <a style="float: right;" href="/articles/page/2">

Next >>>>>

</a>

   </div>

  <hr>

</div>

Criamos em nosso arquivo de view index ao final da pagina um link para a pagina seguinte que ainda não esta dinâmico

* Aula 45 Finalizando a paginação

page.ejs

<div class="row">

<div class="col col-12">

    <% if(result.page > 2){ %>

      <a style="float: lef;" href="/articles/page/<%= result.page - 1 %> ">

          ---- PREV

        </a>

      <% } else if(result.page == 2){ %>

      <a style="float: lef;" href="/"<%= result.page - 1 %> ">

            ---- PREV </a>

      <% } %>

      <% if(result.next){ %>

      <a style="float: right;" href="/articles/page/<%= result.page + 1 %> ">

            NEXT >>>>

         </a>

       <% } %>

   </div>

</div>

Sistema de paginação que mostra a a próxima e a pagina anterior

* Aula 46 Bug na paginação – correção

if(isNaN(page) || page == 0) {

        offset = 0

    } else {

        offset = (parseInt(page) - 1) \* 4

    }

Se atentar a essa situação

Offset, de onde começa a contagem, nesse caso caso a pagina seja 1, a conta sera 1 – 1 \* 4 => 1-1 = 0 => 0 \* 4 = 0, a contagem comecara a partir do zero, caso não tivesse esse menos 1 a contagem começaria no 4 elemento

* Aula 47 Base do sistema de login

UserController.js

const express = require('express')

const router = express.Router()

const user = require('./User')

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

    res.send('Listagem de usuários')

})

router.get('/admin/users/create', (req, res) => {

    res.render('admin/users/create')

})

module.exports = router

user.js

const Sequelize = require('sequelize')

const connection = require('../database/database')

const User = connection.define('users', {

    email: {

        type: Sequelize.STRING,

        allowNull: false

    },

    password: {

        type: Sequelize.STRING,

        allowNull: false

    }

})

Category.sync({force:false})

module.exports = User

criamos um novo model – User e seu controller

* Aula 48 Formulário de cadastro

create.ejs

<form class="form" action="/users/create" method="post">

<input class="form-control" type="email" name="email" placeholder="email@exemplo.com">

   <br>

   <input class="form-control" type="password" name="password" placeholder="senha">

   <br>

   <button class="btn btn-success btn-block" type="submit">Criar</button>

</form>

UserController.js

router.post('/users/create', (req, res) => {

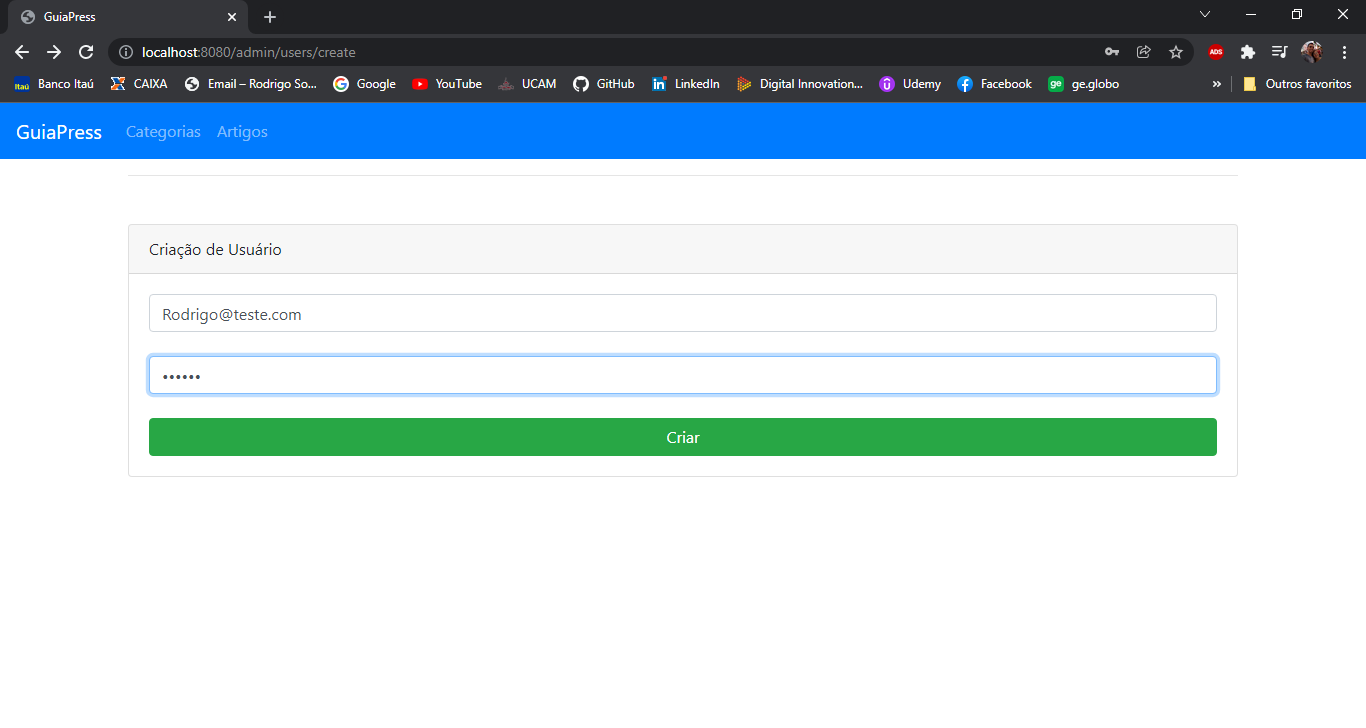
    let email = req.body.email

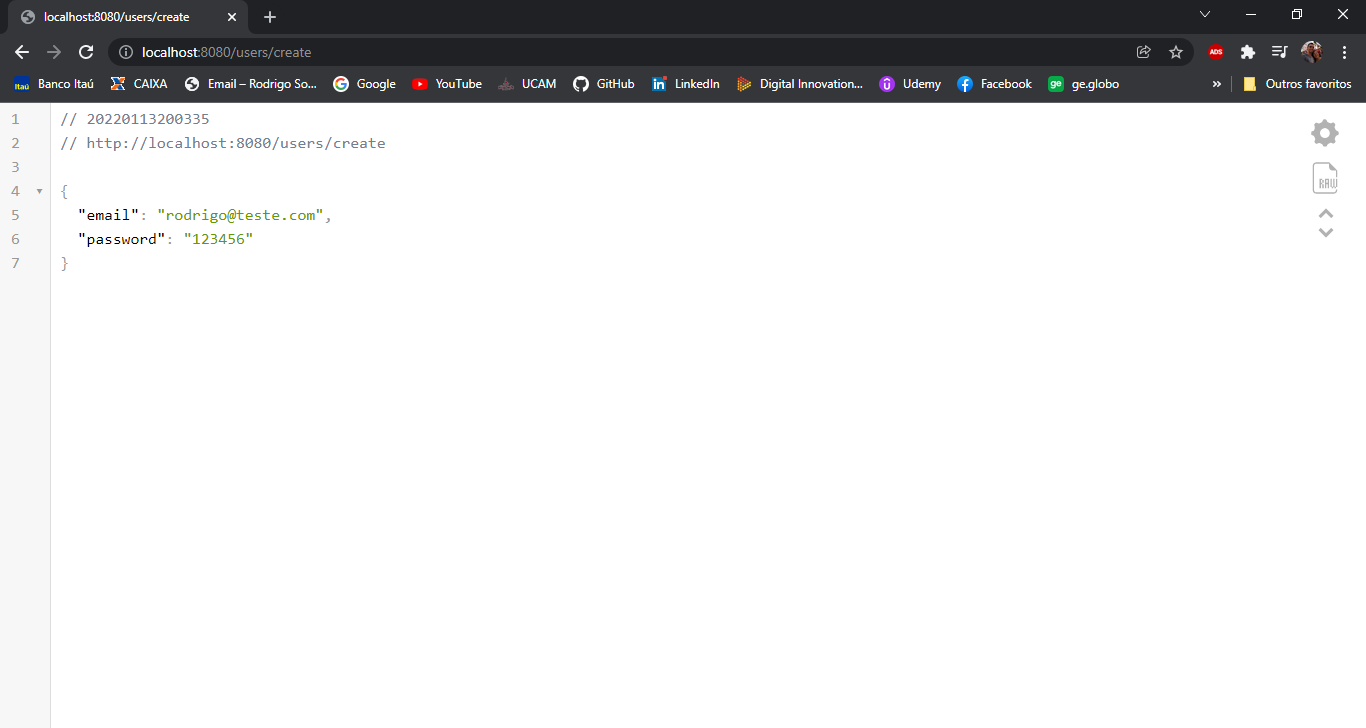
    let password = req.body.password

    res.json({email, password})

})

Testando com o json para ver qual o retorno está nos dando essa rota





Teremos esse retorno com o uso do json de nossa página

DICA:

Para ter essa visualização no chrome usamos a extenção

JSON VIEW

* Aula 49 HASH de senha

NUMCA EM NENHUMA IPOTECE, VOCE DEVE SALVA A SENHA DE USUARIO DA FORMA PLANA(COMO FOI ESCRITA LITERALMENTE) EM SEU BANCO DE DADOS

Por isso iremos usar o modelo HASH para as suas senhas, e quase como um cliptografia, porem não e possível reverter o hash para deciflar a senha como no cliptografia

Para usar esse método iremos usar a bibliote bcryptjs

npm install --save bcryptjs

depois de instalarmos a bibliote bcryptjs, iremos chama la onde iremos usala

UserController.js

const bcrypt = require('bcryptjs')

router.post('/users/create', (req, res) => {

    let email = req.body.email

    let password = req.body.password

    let salt = bcrypt.genSaltSync(10)

    let hash = bcrypt.hashSync(password, salt)

    res.json({email, password})

})

module.exports = router

a variável salt e usada para deixar a nossa senha mais segura

a variável hash será gerado um sequencia de números que irá se tornar a nossa senha, passamos para esse hash a nossa variável password e nosso salt, para deixala mais segura

UserController.js

router.post('/users/create', (req, res) => {

    let email = req.body.email

    let password = req.body.password

    let salt = bcrypt.genSaltSync(10)

    let hash = bcrypt.hashSync(password, salt)

    User.create({

        email:email,

        password:  hash

    }).then(() => {

        res.redirect('/')

    }).catch(err => {

        res.redirect('/')

    })

})

Agora estamos pegando o model de User e fazendo o create nele, passando os valores email e password e caso tudo ocorra bem estamos redirecionando para a tela principal

* Aula 50 Impedindo email duplicado

UserController.js

router.post('/users/create', (req, res) => {

    let email = req.body.email

    let password = req.body.password

    User.findOne({ // Procura no banco, onde email seja igual ao email passado nos campos do model de User

        where:{

            email: email

        }

    }).then(user => { // O valor da pesquisa será enviado para user

        if(user == undefined){  // Caso NÃO exista

            let salt = bcrypt.genSaltSync(10)

            let hash = bcrypt.hashSync(password, salt)

            User.create({

                email:email,

                password:  hash

            }).then(() => {

                res.redirect('/')

            }).catch(err => {

                res.redirect('/')

            })

        } else {                 // Caso EXISTA

            res.redirect('/admin/users/create')

        }

    })

})

Modelo que verifica se o campo email já existe em nosso banco

Com o uso do findOne, que procura pelo campo email e compara com o email passado no body

* Aula 51 Listagem de usuários

UserController.js

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

    User.findAll().then(users => {

        res.render('admin/users/index', {users: users})

    })

})

Rota em questão que faz uma pequisa em todo o banco do model User, que renderiza uma pagina e envia esses campos que foram pesquisados

user/index.ejs

<tbody>

            <% users.forEach(user => { %>

                <tr>

                    <td> <%= user.id %> </td>

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

                </tr>

            <% }) %>

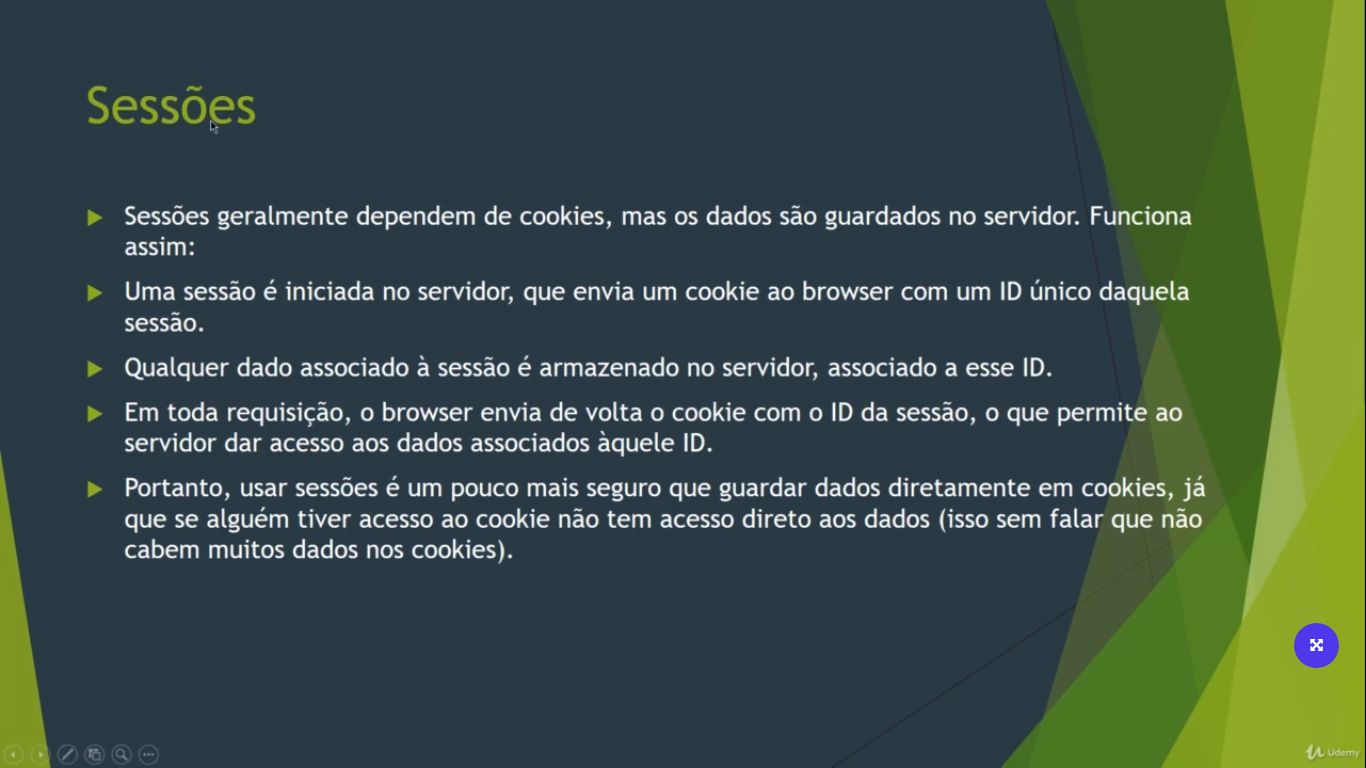
</tbody>

Pagina que ira receber o parâmetro, para cada volta dentro de users, ele ira criar uma tabela com o campo presente em user.id e user.email

* Aula 52 O que são cookies e sessões







* Aula 53 Express sessions

Vamos utilizar as sessões para fazer o nosso sistema de login, que irá funcionar da seguinte maneira:

Vai ter o formulário de login, o usuário vai digitar o login dele e caso o login esteja correto nos iremos gerar uma sessão para ele, e com essa sessão gerada ele ira poder acessar todas as rotas do painel administrativo, basicamente a nossa regra de altenticação será a seguinte, quem tiver a sessão de login, pode acessar todas as rotas do admin e quem não tiver não conegue acessar nenhuma rota do admin.

Para trabalhar com sessoes no express, vamos trabalhar com uma biblioteca chamada de express-session

npm install express-session

agora so configurar essa biblioteca

index.js

const session = require('express-session')

após chama-la em nosso arquivo principal, temos que ativar o gerenciamento de sessões da nossa aplicação

index.js

// Ativar gerenciamento de sessão

app.use(session({

    secret: 'qualquercoisa',

    cookie: {

        maxAge: 30000

    }

}))

Uma palavra qualquer que o express-session ira utilizar para almentar a segurança das suas sessões, como se fosse uma senha para decriptar suas sessões, pode se colocar qualquer coisa porem e recomendado colocar coisas aleatórias, como se fosse o salt do bcrypt

ex:

secret: 'qdiwehnfcebthuirbybim',

além disso precisamos colocar como o cookie sera salvo no navegador do usuário

maxAge -> define o tempo máximo que um usuário poderá ficar logado ate ser deslogado altomaticamente

nesse caso o valor é em milissegundo

1 segundo = 1000 milissegundos

60 segundo = 60000 milissegundos

Para que a sessão seja fechada assim que o naegador for fechado basta não declara esse atributo

* Aula 54 Manipulando sessões

Sessões basicamente só tem dois processo, escrita e leitura, você pode salvar qualquer dado em suas sessões, que esse dados irão ficar salvo globalmente no seu aplicativo, você conseguirar acessar essa dado em qualquer rota do seu aplicativo, sessão e algo global para a sua aplicação

Você pode criar quantas informações de sessões quiser, e para criar basta seguir esse passo:

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

    req.session.treinamento = 'Formação Node.js'

})

Nome do campo que quer criar

Valor do campo criado

index.js

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

    req.session.treinamento = 'Formação Node.js'

    req.session.ano = 2019

    req.session.email = 'victor@udemy.com'

    req.session.user = {

        username: 'Rodrigo',

        email: 'rodrigo\_souza.91@hotmail.com',

        id: 10

    }

 res.send('Sessão gerada!!!')

})

Toda a vez que essa rota for acessada /session, esses dados serão salvo em sessão para mim na aplicação, isso que dizer que apartir do momento que esse código e gerado, ele poderar ser acessado em qualquer parte da aplicação

index.js

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

    req.session.treinamento

})

Aqui estamos acessando a sessão treinamento criada na rota /session

Toda a rota precisa retornar um resposta, então iremos mostrar uma json dos campos da session

index.js

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

    res.json({

        treinamento: req.session.treinamento,

        ano: req.session.ano,

        email: req.session.email,

        user: req.session.user

    })

})

ATENÇÃO: Essa rota /leitura, só retornará valores depois que a rota /session for iniciada uma vez, levando em conta o tempo que estive no maxAge, para apagar a sessão(ao reiniciar o servido as sessões são perdidas), aqui a session e zerada a cada 30 segundo

cookie: {

        maxAge: 30000

    }

Para trocar essa opção do tempo para deixar para apagar a session somente quando o navegador fechar basta colocar o código:

req.session.cookie.expires = false

DETALHE: O express-session, tem o que chamamos de storage( e onde a sessão ficara salva sem seu sistema), por padrão o express-session salva suas sessões na memoria do servido, no caso está salvando na memoria ram do computador

Isso acaba sendo um problema em aplicação de médio e longa escala, porque não e nem um pouco legal salvar suas sessões na memoria, porque em um sistema assim que tem muitas requisição ao longo do tempo, logo chegara uma hora que sua memoria ira estrourar pois não haverá mais espaço para salvar tantas sessões

Para resolver podemos mudar o storage do express-session, com o uso do banco de dados REDIS (banco apropriado para esse tipo de salvamento de chash) em vez do storoge padrão

* Aula 55 Autenticação

users/login.ejs

<%- include ('../../partials/header')%>

<%- include ('../../partials/navbar') %>

<div class="container">

    <hr>

    <div class="card mt-5">

        <div class="card-header">

            <h2>Login de admin</h2>

        </div>

        <div class="card-body">

            <form class="form" action="/authenticate" method="post">

                <input class="form-control" type="email" name="email" placeholder="email@exemplo.com">

                <br>

                <input class="form-control" type="password" name="password" placeholder="senha">

                <br>

                <button class="btn btn-success btn-block" type="submit">Logar</button>

            </form>

        </div>

    </div>

</div>

<%- include ('../../partials/footer') %>

Temos um form que está apontando para a rota /authenticate

Usercontroller.js

router.post('/authenticate', (req, res) => {

let email = req.body.email

   let password = req.body.password

   User.findOne({where: {email: email}}).then(user => {

     if(user != undefined){ // SE EXISTE usuario com esse email

        //Validar senha

        let correct = bcrypt.compareSync(password, user.password)

        if(correct){ // Senha iguais

        req.session.user = {

            id: user.id,

              email: user.email

              }

                res.json(req.session.user)

        } else {

                res.redirect('/login')

            }

     } else {

         res.redirect('/login')

        }

    })

})

Nessa rota recebemos dois valores, o email e o password do formulário enviado

Depois colocamos para procurar no model User, uma linha, onde o email senha igual ao email passado pelo formulário e jogamos essa informação na variável user.

caso exista, ele cria uma variável correct, que verifica se o password passado no formulário e o mesmo valor do user.password, com o uso do bcrypt.compareSync

caso o password seja igual, ai sim ele cria uma sessão com o campo user, contendo os valores de user.id e user.email.

caso alguma da busca não retorne nada, a pagina sera redirecionada para a pagina de login /login

agora que temos a autenticação sendofeita, so falta a parte de autorização, seria a parte de falar se o usuário pode acessar determinada rota ou não

* Aula 56 O que é um Middleware

Func() {

Console.log(‘OI’)

}

RES

REQ

MIDDLEWARE

GET(‘Olá’)

Usuario mando uma requisição e a rota mandando uma resposta ao usuário, o middleware e algo que fica no meio dessa duas coisas,

A cada momento que o cliente faz uma requisição para o rota, essa requisição passa pelo middleware, que e possível processala, e aparti do momento que a rota retorna uma resporta, essa resposta passa por dentro do middleware

Nesse exemplo temos o middleware, que toda a vez que o usuário acessa a requisição da rota e printado no console a palavra oi, e toda a vez que e enviada uma resposta dessa rota e printado no console a palavra oi, as rotas que tiverem esse middleware será mostrado essa ação

* Aula 57 Autorização com middleware

Agora eu so quero permitir que usuarios que tenha a sessão user, que foi criado na aula 55, possam acessar a rota do painel administrativo e impedir quem não tenha essa sessão de acessa essa rota.

Para fazer isso vamos colocar uma condição em nossas rotas

UserController.js

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

    if(req.session.user == undefined){ //Não existe a sessão user

        res.redirect('/')

    }

    User.findAll().then(users => {

        res.render('admin/users/index', {users: users})

    })

})

Essa condição verifica se existe uma sessão chamada de user, caso não exista ela entrar na condição, que ira redirecionar a pagina para a pagina principal, e caso exista uma sessão user ela ira continuar o código fazendo seu caminho normal

Agora para cada rota que você queira que seja feita essa autorização, basta fazer essa checagem

Logo teremos um ploblema, se tivermos 100 rotas, teríamos que colocar essa checagem 100 vezes, e caso nosso método de autenticação mude, teremos que refazer todo o código,(apagar o código antigo e colocar o novo)

Para resolver essa questão ultilizaremos o middleware

Iremos plugar esse middleware na rota

Para criar uma middleware(que e uma função normal) basta, criar uma pasta de middlewares(poderia ser criada em qualquer lugar porem para manter a organização vamos criar uma pasta) e nessa pasta vamos criar um arquivo adminAuth.js

adminAuth.js

function adminAuth(req, res, next) {

    if(req.session.user != undefined){

        next()

    } else{

        res.redirect('/login')

    }

}

module.exports = adminAuth

Todo middleware tem três parâmetro, a requisição a resposta e o next que envia a rota para o próximo passo, como vimos no desenho da aula 56, caso não tenha o next a aplicação ficara parada

Depois de exportada basta colocar no controller que precisar desse middleware

UserController.js

// Importando um MIDDLEWARE

const  adminAuth = require('../middlewares/adminAuth')

depois de importar esse middleware basta plugar em cada rota que precisar de autenticação

UserController.js

// Importando um MIDDLEWARE

const  adminAuth = require('../middlewares/adminAuth')

router.get('/admin/articles', adminAuth, (req, res) => {

    Article.findAll({

        include:[{model:Category}]

    }).then(articles => {

        res.render('admin/articles/index', {articles: articles})

    })

})

Assim que se coloca o middleware em uma rota

Agora essa rota so sera acessada caso tenha a sessão user, caso contrario sera redirecionada para a pagina de login

A rota só sera acessada caso o usuário tenha feito o login para gerar a sessão, apartir da rota /login, que ira apresentar uma pagina com um formulário que ao logar envia para a rota /authenticate que ira autenticar o login para criar a sessão e assim ter acesso ao restante da pagina a não ser as paginas que qualquer usuário possa ter acesso que não precise de autenticação

* Aula 58 Logout

O que define se um usuário esta logado e se ele tem ou não a sessão user

Para apagar essa sessão user iremos deslogar o usuário

UserController.js

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

    req.session.user = undefined // deixar o user nulo

    res.redirect('/')

})

Criamos a rota que deixa a sessão user como undefined