Modulo 07

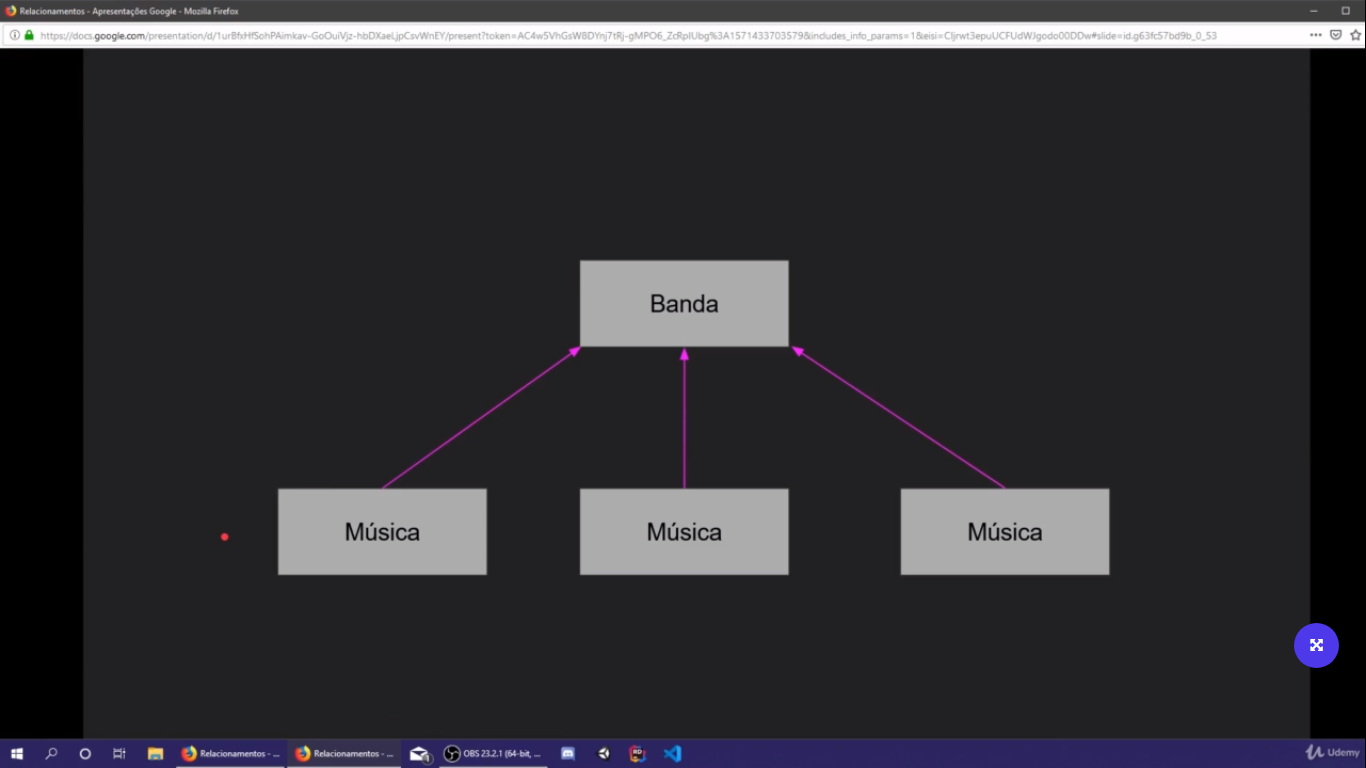
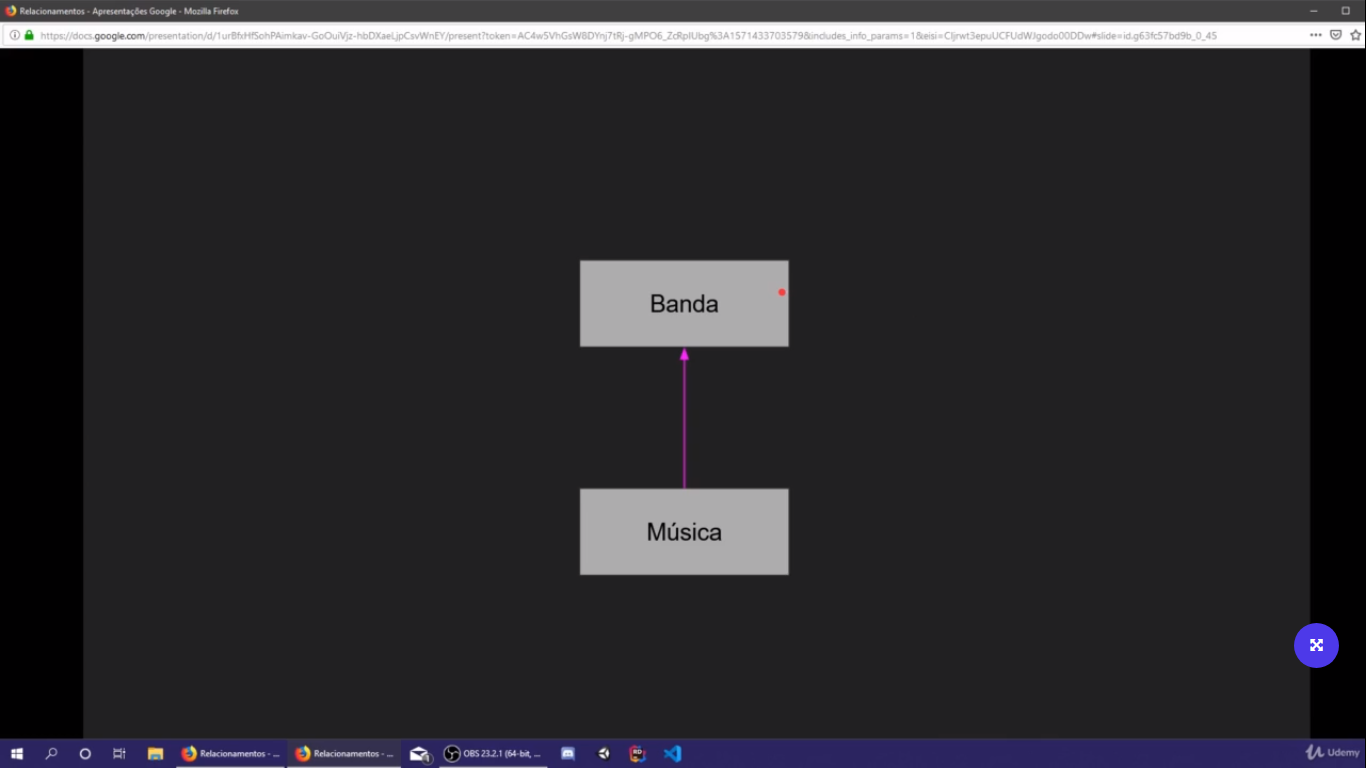
Projeto 02 Criando um blog com painel administrativo – CRUD completo

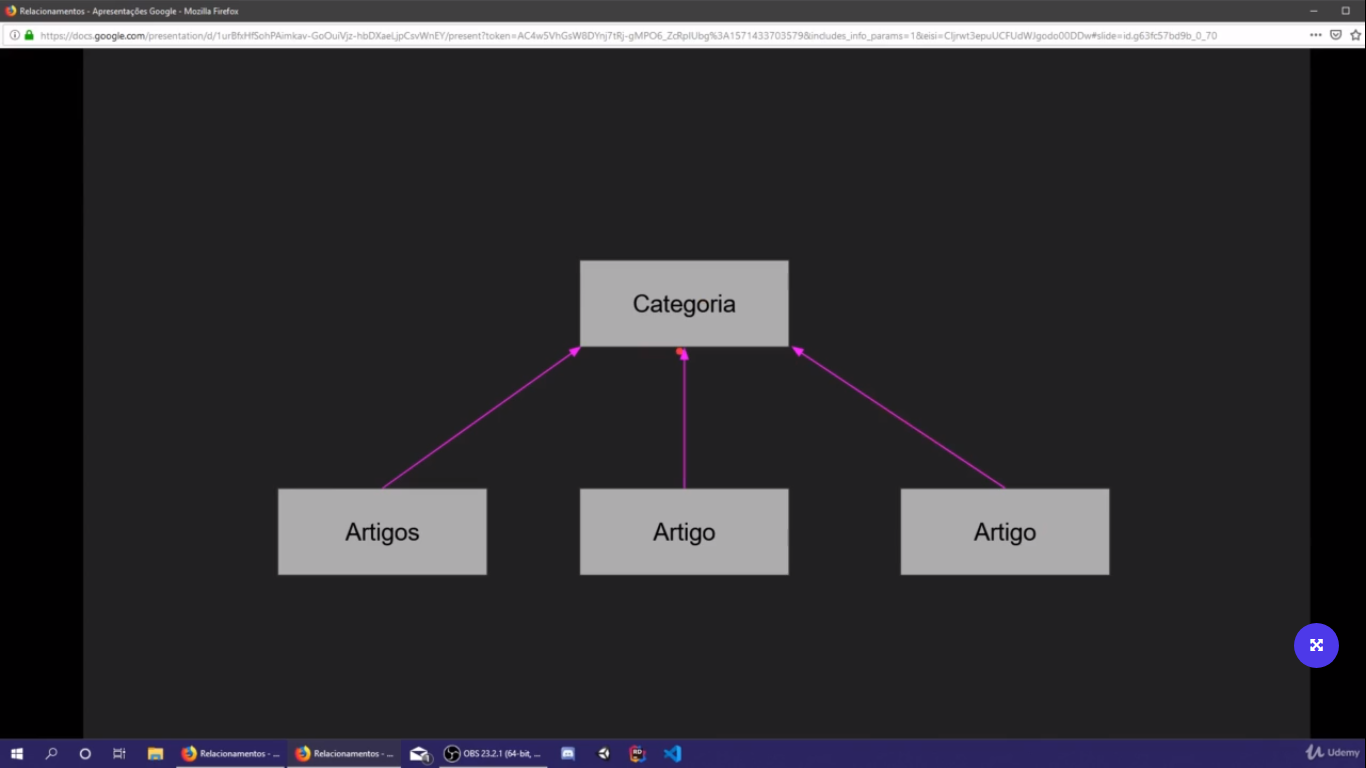
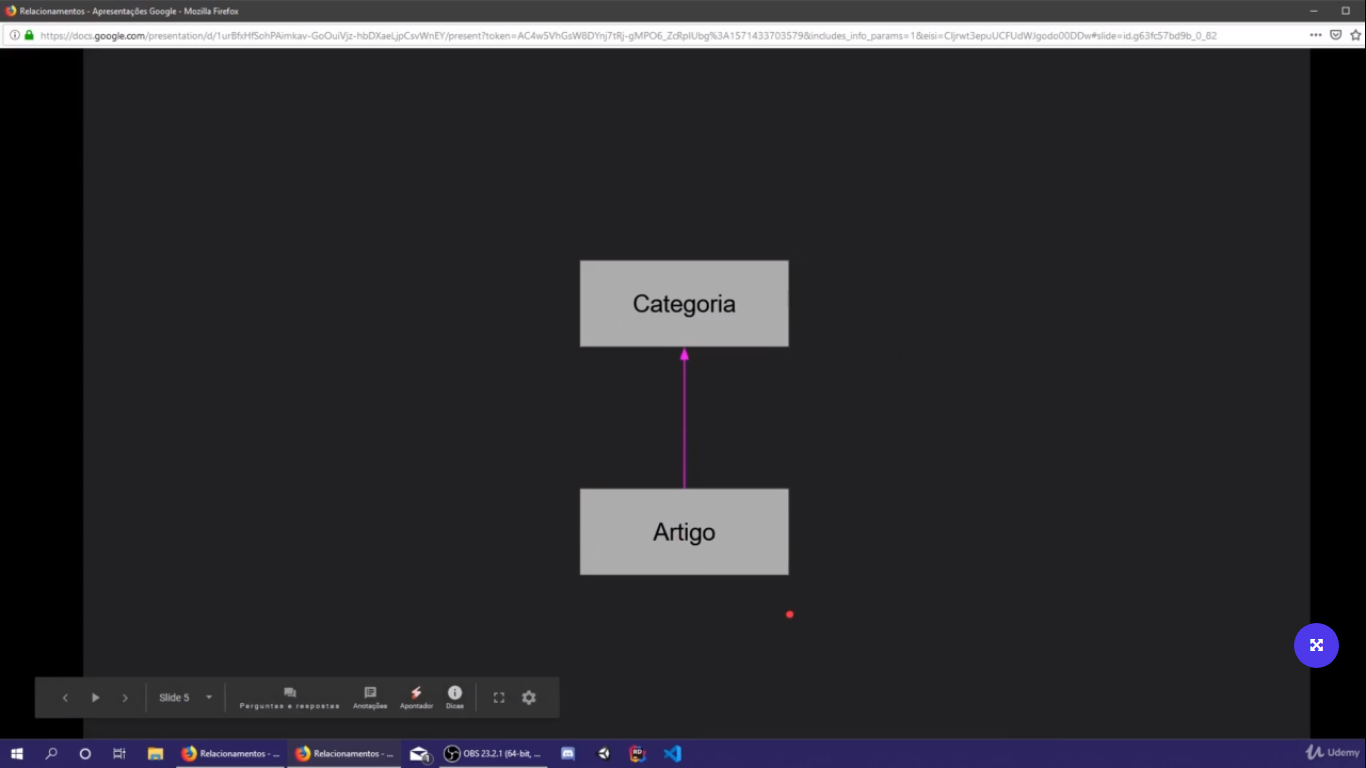
* Aula 01 Introdução

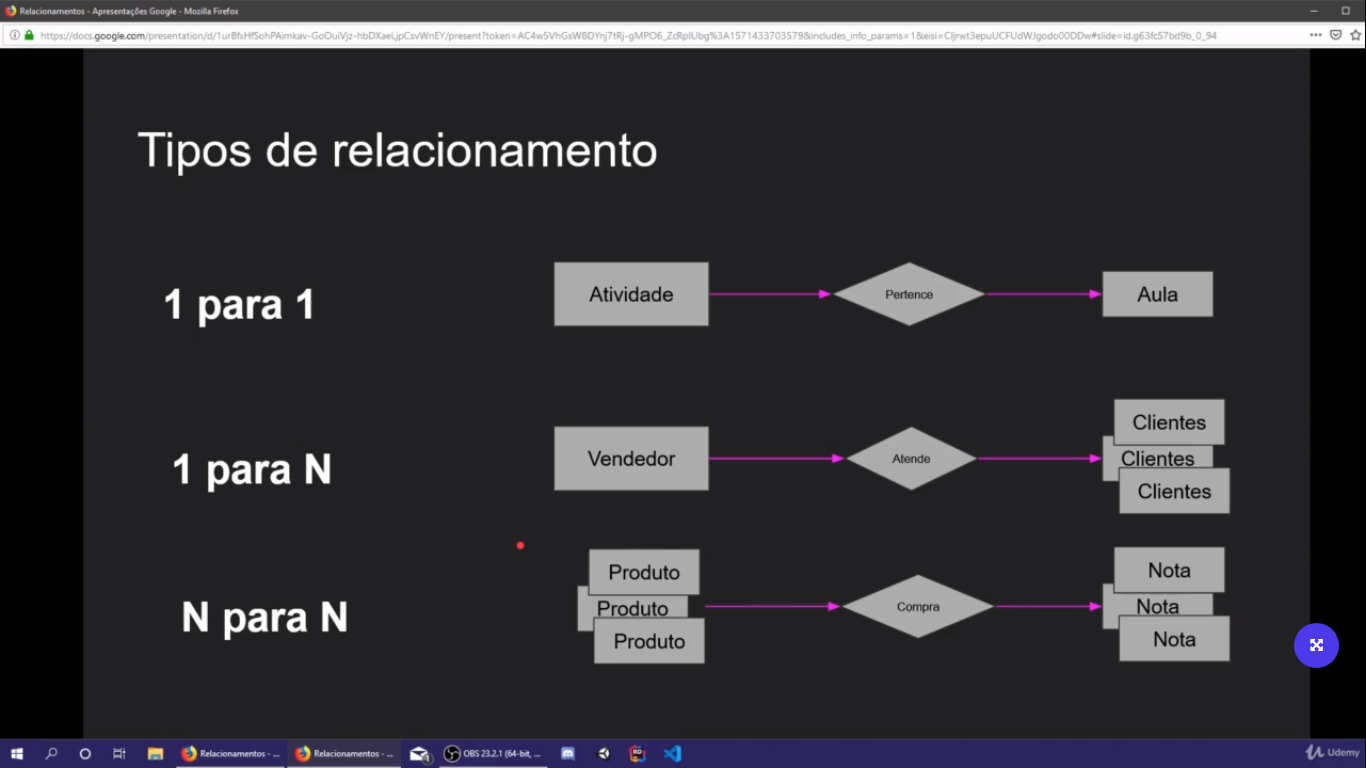
TinyMCE – Editor de texto mais utilizado para formatação de texto do estilo de blogs

* 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

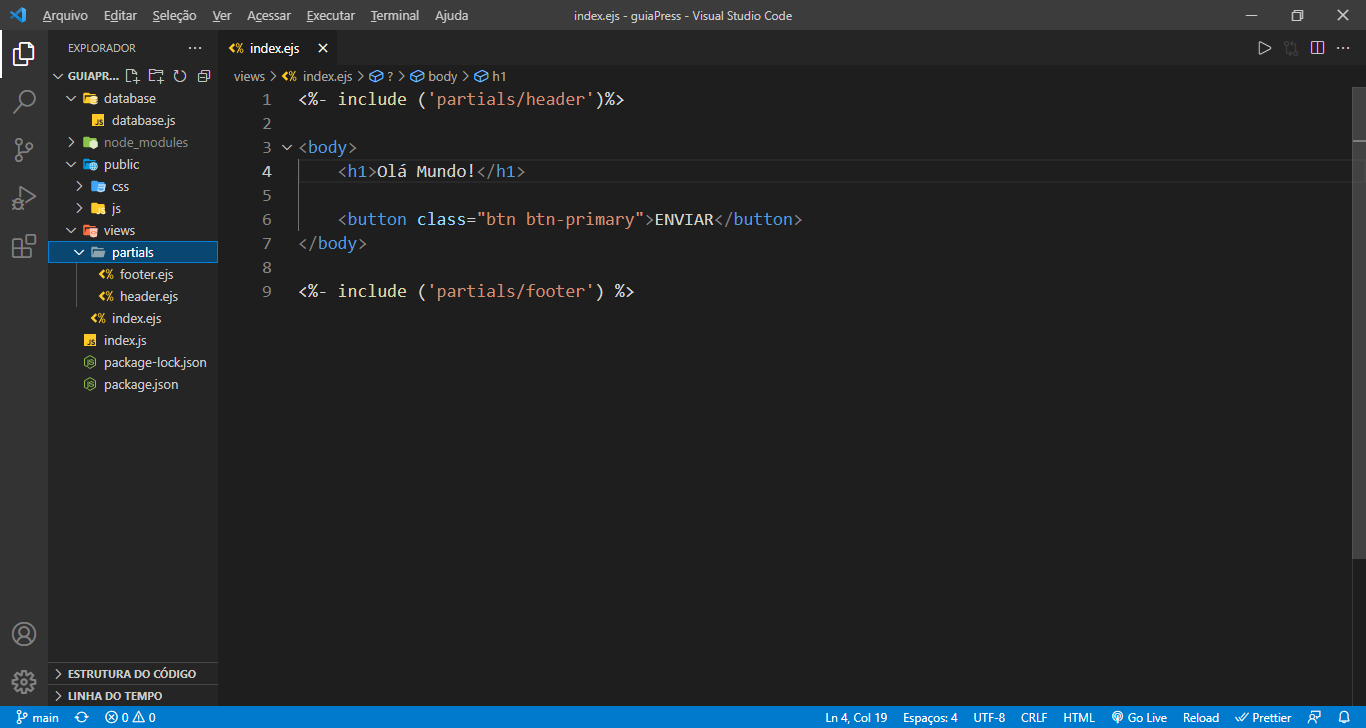
Feito essas configurações iremos instalar o bootstrap no projeto

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

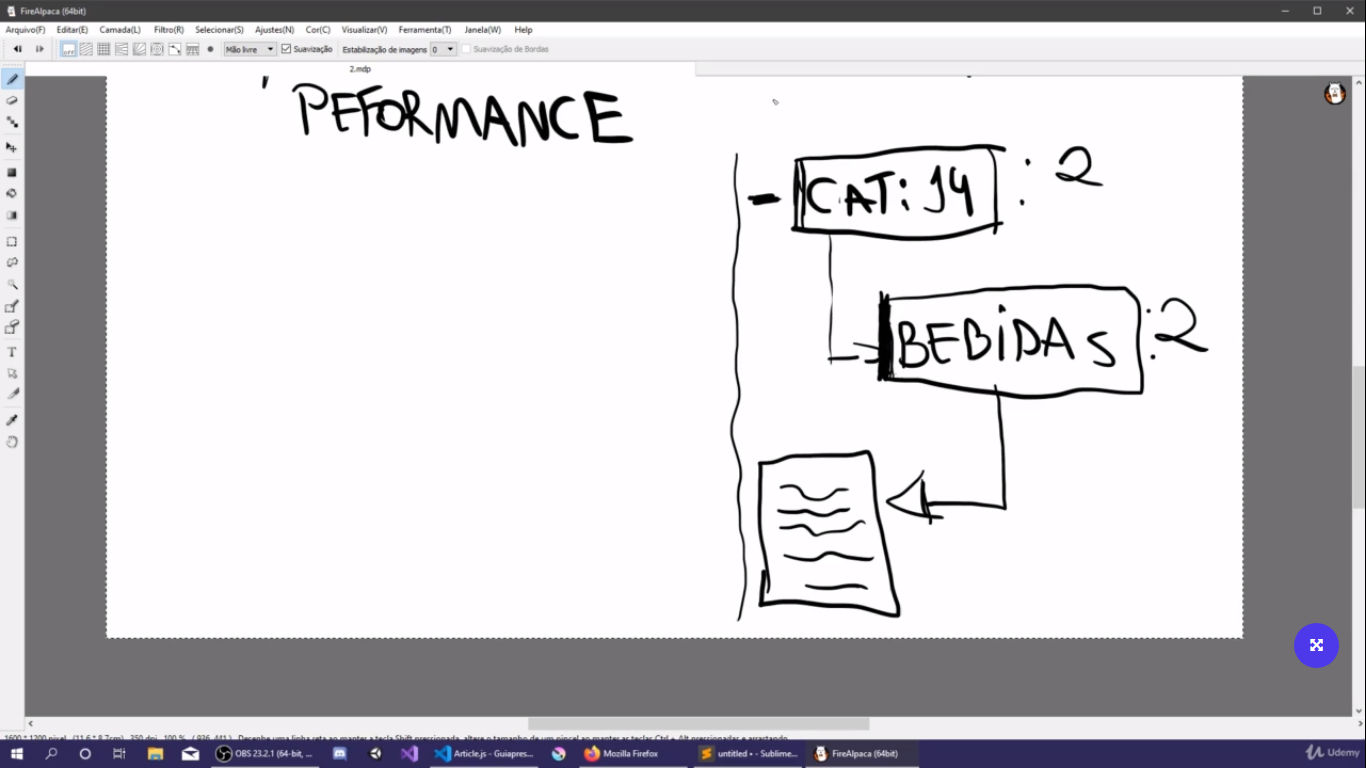
//Category.sync({force:true}) APAGAR

Article.js

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

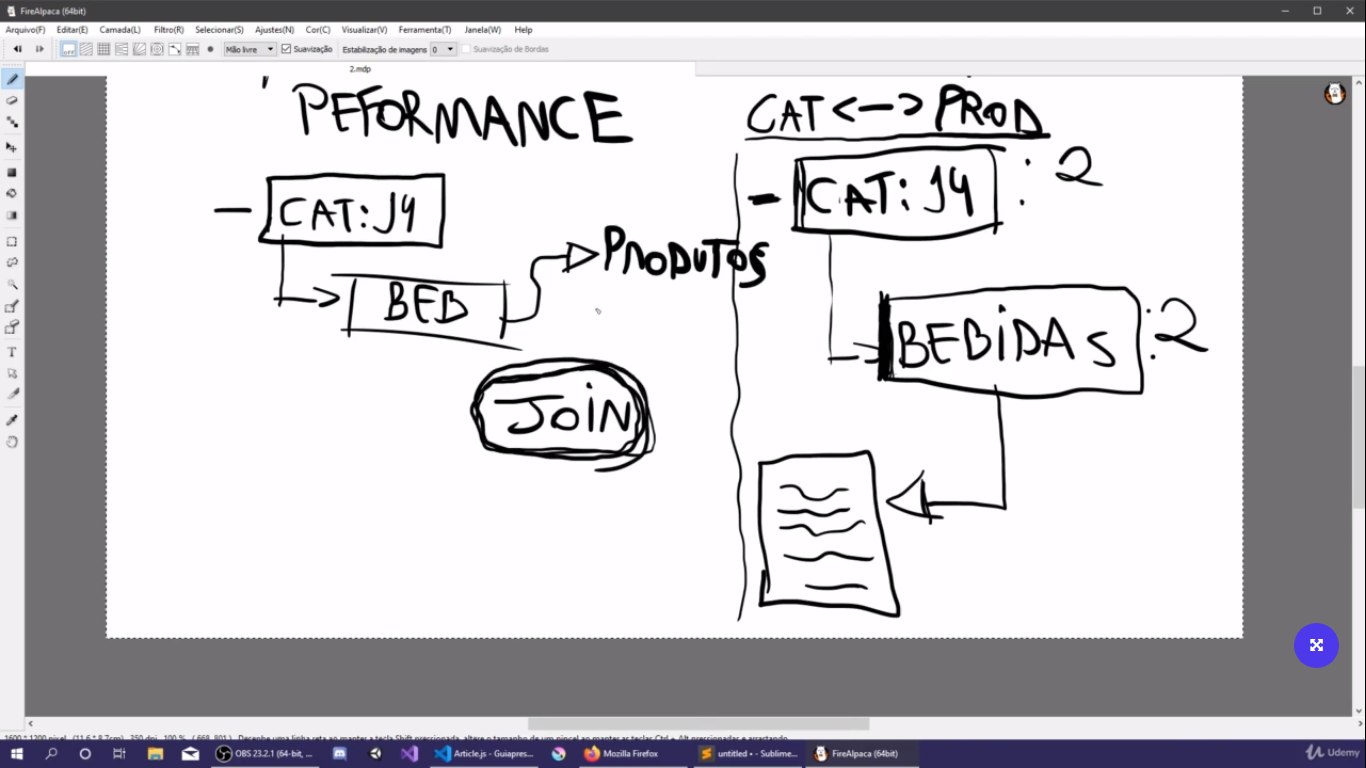
* Aula 09 qual a utilidade dos relacionamentos?

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="/categories/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="/categories/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="/categories/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