PROJETO PARA CONSTRUIR UM BLOG COM EDITOR DE TEXTO ÁREA DE LOGIN E PAINEL ADMINISTRATIVO

1. Tipos de relacionamentos :
   1. - 1 para 1 – Uma atividade pertence a uma Aula.
   2. - 1 para N – Um vendedor atende N clientes
   3. - N para N – Um produtos está em varias notas, Uma nota possui vários produtos,
2. Base do projeto
3. Vamos instalar todas as bibliotes, como o express, o body-parser, o sequelize e o ejs.
4. Vamos configurar o projeto com bootstrap, vamos criar o layout inicial;
5. Cria a pasta do projeto, ‘GuiaPress’

NO TERMINAL

1. Entrar na pasta do projeto
2. Digitar ‘npm init’ para iniciar a estrutura do projeto
3. Digitar ‘npm install –save express’
4. Digitar ‘npm install –save sequelize’
5. Digitar ‘npm install –save mysql2’ – dependência do sequelize para se comunicar com o mysql.
6. Digitar ‘npm install –save body-parser’ – biblioteca do express para trabalhar com formulários
7. Digitar ‘npm install –save ejs’ – que é a template engine para renderizar html

NA PASTA DO PROJETO

1. Criar o arquivo principal index.js

ABRIR O ARQUIVO index.js

1. Importar o express para dentro do index.js – ‘const express = require(‘express’);’
2. Criar uma instância do express – ‘const exp = express();
3. Criar a rota principal – ‘app.get(‘/’,(req, res)=>{res.sender(“Bem vindo ao meu site ’)};
4. Iniciando a aplicação – ‘app.listen(8080,()=>{ console.log(“O servidor está rodando)});
5. Testar a aplicação com – ‘nodemon index.js’;
6. Aapós a crianção da instância do express, configurar a engine ejs - ‘app.set(‘view engine’, ‘ejs’);’

NA PASTA DO PROJETO

1. Criar a pasta Views

NA PASTA VIEWS

1. Criar o arquivo ‘index.ejs’ – primeira view do projeto

ABRIR O ARQUIVO INDEX.JS

1. Após a configuração da view engine, configurar também o express.static para configurar a pasta dos arquivos estáticos(css, img, js)
2. ‘app.use(express.static(‘public’));

NA PASTA DO PROJETO

1. Criar a pasta public

ABRIR O MYSQLWORKBECH

1. Criar o banco de dados chamado ‘guiapress’

NA PASTA DO PROJETO

1. Criar a pasta database

NA PASTA database

1. Criar o arquivo database.js
2. Importar o sequelize – ‘const Sequelize = require(‘selequelize);’
3. Criar a conexão com o banco de dados – ‘const connection = new Sequelize(‘guiapress’,’root’,’123456’,{ host: ‘localhost’, dialect:’mysql’});
4. Exportar a conexão – ‘module exports = connection;’

NO ARQUIVO INDEX.JS

1. Importar a conexão do banco de dados – ‘const connection = require(‘./database/database’);
2. Realizar a autenticação com o banco de dados – ‘connection.authenticate().then(()=>{ console.log(“conexão feita com sucesso!”).catch((error)=>{console.log(error)});

NA PÁGINA DO BOOTSTRAP

1. Baixar o bootstrap

NA PASTA public

1. Colar conteúdo do bootstrap

NA PASTA views

1. Criar a pasta partials

NA PASTA partials

1. Criar o arquivo header.ejs,

NO ARQUIVO header.ejs

1. Digitar
2. <!DOCTYPE html>
3. <html lang="pt-br">
4. <head>
5. <meta charset="UTF-8">
6. <meta http-equiv="X-UA-Compatible" content="IE=edge">
7. <meta name="viewport" content="width=device-width, initial-scale=1.0">
8. <title>GuiaPress</title>
9. <link rel="stylesheet" href="/css/bootstrap.min.css">
10. </head>
11. <body>

NA PASTA partials



1. Criar o arquivo footer.ejs

NO ARQUIVO footer.ejs

1. Digitar
2. </body>
3. <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>
4. <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js" integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7lN" crossorigin="anonymous"></script>
5. <script src="/js/bootstrap.min.js"></script>
6. </html>

NA PASTA views

1. Criar o arquivo principal index.ejs

NO ARQUIVO index.ejs

1. Incluir o header da página
2. Digitando <%- include ('partials/header.ejs'); %>
3. Incluir o footer da página
4. Digitando <%- include ('partials/footer.ejs'); %>
5. OBS: O conteúdo da página deve estar localizada entre o header e o footer
6. Para melhorar a arquitetura e organização do projeto vamos criar várias pastas para separar a lógica do programa por assuntos no formato MVC

NA PASTA DO PROJETO

1. Criar as pastas categories e articles

NA PASTA categories

1. Criar o arquivo categoriesController.js

NO ARQUIVO categoriesController

1. Importar o express – ‘const express = require(‘express’);’
2. Carregar o router que será um objeto para carregar as rotas
3. Const router = express.Router();
4. Criar a rota categories
5. Digite router.get(‘categories’,(req,res)=>{ res.send(‘Rota de categorias’)});
6. Criar a rota para criar nova categoria
7. Digite router.get(‘/admin/categories/new’,(req,res)=>{res.send(‘Rota para criar categorias)});
8. Exportar o arquivo do router para que ele possa ser utilizado no arquivo principal
9. Digite module.exports = router;

NO ARQUIVO index.js

1. Importar o router
2. Digite ‘const categoriesController = require(‘./categoriesController/CategoriesControlle’);
3. Digite ‘app.use(‘/’,categoriesController);’ para o aplicativo poder utilizar o conteúdo do controller CategoriesController; e a rota funcionar.
4. O ‘/’ anterior serve somente como prefixo ou seja para se acessar a rota de categorias é necessário antes colocar o prefixo. Poderia ser outro qualquer.

NA PASTA articles

1. Criar outro arquivo com o nome ArticleController que conterá as rotas dos articles

NO ARQUIVO ArticleController

1. Copiar todo conteúdo do controller CategoriesController e colar neste arquivo em branco
2. Alterar as rotas substituindo por articles

NA PASTA categories

1. Criar o arquivo Category.js que será o model de categorias

NO ARQUIVO Category.js

1. Importar o sequelize – ‘const Sequelize = require(‘selequelize’);
2. Importar a conexão com o banco de dados
3. Digitar ‘const connection = require(‘../database/database’);
4. Definir o model com os campos que irão compor a tabela
5. Const Category = connection.define(‘categories’, {
6. Tilte:{
   * 1. Type: Sequelize.STRING,
     2. allowNull: false}, slug:{ type: Selequize.TEXT, allowNull: false}});
7. Criando a tabela
8. Digite ‘Article.sync({force:true});
9. OBS: Deve-se comentar ‘Article.sync após a tabela ser criada no banco de dados para não ocorrer erros
10. Exportando o model
11. Digite module.exports = Category;

NA PASTA articles

1. Criar o arquivo Article.js que será o model da tabela articles

NO ARQUIVO Article

1. Importar o sequelize – ‘const Sequelize = require(‘selequelize’);
2. Importar a conexão com o banco de dados
3. Digitar ‘const connection = require(‘../database/database’);
4. Importar o outro model que fará parte do relacionamento
5. Const Category = require(‘./categories/categories’);
6. Definir o model com os campos que irão compor a tabela
7. Const Article = connection.define(‘articles, {
8. Tilte:{
   * 1. Type: Sequelize.STRING,
     2. allowNull: false}, slug:{ type: Selequize.TEXT, allowNull: false},
     3. body:{ type:Sequelize.TEXT, allowNull:false}});
9. Definindo os relacionamento
10. Relacionamento um para um
11. Digite ‘Article.belongsTo(Category);’
12. Relacionamento um para muitos
13. Digite ‘Category.hasMany(Article);
14. Criando a tabela, após criar a tabela no banco de dados, deve-se comentar a linha abaixo
15. Digite ‘Article.sync({force:true});
16. Exportando o model
17. Digite module.exports = Article;

NO ARQUIVO index.js

1. Importar os models
2. ‘const Article = require(‘./articles/Article’);’
3. ‘const Category = require(‘./categories/Category’);’

08/08/21

NA PASTA views

1. Criar uma pasta chamada admin que terá todas as páginas referentes a cadastros e gerenciamento do site.

NA PASTA admin

1. Cria uma pasta ‘categories’ para armazenar todas as páginas de gerenciamento de categorias

NA PASTA categories

1. Criar o arquivo new.ejs que será a página para cadastro das categorias

NO ARQUIVO new.ejs

1. Incluir os arquivos partials digite:
2. <%- include(‘../../partials/header’);
3. <%- include(‘../../partials/footer’);
4. OBS: todo conteúdo da página deve estar entre o header e o footer

NO ARQUIVO categorieController

1. Alterar a rota que foi criada para acessar a página new
2. Digitar dentro da rota ‘res.render(‘admin/categories/new’);
3. Dentro do método render deve-se digitar todo o endereço quando o arquivo não estiver em view.

NA PASTA partials

1. Criar o arquivo navbar.ejs que será a navbar das páginas

NO ARQUIVO navbar.ejs

1. Digitar
2. <nav class="navbar navbar-light bg-primary">
3. <a class="navbar-brand" href="/">
4. GuiaPress
5. </a>
6. </nav>

NO ARQUIVO index.ejs

1. Incluir a nav bar após o header da página
2. <%- include ('partials/navbar.ejs'); %>

NO ARQUIVO new.ejs

1. Incluir a navbar após o header da página
2. <%- include ('../../partials/header.ejs'); %>

NO ARQUIVO categoriesController

1. Criar a rota para receber os dados do formulário da pagina new e salvar na tabela categorie.
2. Digite
3. //criando a rota para receber os dados do formulário e salvar na tabela categories
4. router.get('/categories/save',(req, res)=>{
5. //pegar os dados do formulário criando uma variável que
6. //irá armazenar o valor do campo input 'title'
7. var title = req.body.title;
8. //verificar se o título existe
9. if(title != undefined){
11. }else{
12. //redirecionando o usuário para a página news pois ele não
13. //o campo de title está vazio
14. res.redirect('/admin/categories/new');
15. }
16. });
17. Importar o modulo de categoria para acesso a tabela categories digite
18. const category = require('../categories/Category');
19. instalar a bibliote Slugfy do node para salvar o slug, que será o título sem espaço ou ponto. As palavras serão separadas por um ‘ – ‘;

NO TERMINAL

1. Digite ‘npm install –save slugify’

NO ARQUIVO categoriesController

1. Carregar o slugfy digite no topo:
2. ‘const slugify = require(‘slugigy’);’
3. Após if(title !=undefined) digite
4. ‘Category.create({
5. title: title,
6. //slug é a versão otimizada para url utilizando a bibliteca
7. // do node chamada slugify
8. slug: slugify(title)
9. });
10. Para capturar dados de formulários é necessário realizar a configuração do express digite
11. //utilizando o express.urlencoded para trabalhar com formulários
12. router.use(express.urlencoded({extended:false}));
13. //configuracao para utilizar os dados do formulário em json
14. router.use(express.json());
15. Criando rota para realizar consulta à tabela categories
16. Digite
17. /criando rota para pegar todos os dados da tabela categories e
18. //mostrá-los na página index.ejs da pasta categorias
19. router.get('/admin/categories',(req,res)=>{
20. //busca todos os registros sem filtro 'where'
21. Category.findAll({
22. raw: true,
23. order:[['title','ASC']]
24. }).then(categories =>{
25. res.render('admin/categories/index',{categories: categories})
26. });
27. });

NA PASTA views/categories

1. Crie o arquivo index.ejs

NO ARQUIVO views/categories/index.ejs

1. Faça os includes do header, navbar e footer
2. Construa a tabela conforme abaixo:
3. <div class="container">
4. <hr>
5. <h2>Categorias</h2>
6. <hr>
7. <table class="table table-bordered">
8. <thead>
9. <tr>
10. <th>Id</th>
11. <th>Título</th>
12. <th>Slug</th>
13. <th>Ações</th>
14. </tr>
15. </thead>
16. <tbody>
17. <% categories.forEach(category=>{ %>
18. <tr>
19. <th><%=category.id %></th>
20. <th><%=category.title %></th>
21. <th><%=category.slug %></th>
22. <th></th>
23. </tr>
24. <%} ) %>
25. </tbody>
26. </table>
27. </div>

NO ARQUIVO navbar.ejs na pasta views/partials

1. Digite
2. <!--Para criar um menu utilizamos a class = navbar-nav
3. mr-auto serve para configurar uma margem total à direita
4. -->
5. <ul class="navbar-nav mr-auto ">
6. <!--Para criar um item  utilizamos a class = nav-item -->
7. <li class="nav-item">
8. <!--O link do item do menu deve ter a class = nav-link
9. para aparecer no menu -->
10. <a class="nav-link" href="/admin/categories">Categorias</a>
11. </li>
12. </ul>

NO ARQUIVO views/categories/index.ejs

1. No arquivo altere o texto do forEach conforme abaixo
2. <% categories.forEach(category=>{ %>
3. <tr>
4. <th><%=category.id %></th>
5. <th><%=category.title %></th>
6. <th><%=category.slug %></th>
7. <th>
8. <a class="btn btn-warning" href="">Editar</a>
9. <a class="btn btn-danger" href="" >Excluir</a>
10. </th>
11. </tr>
12. <%} ) %>
13. Incluir o botão de criar nova categoria abaixo do <hr> do topo da página digite:
14. <a class="btn btn-success" href="../../admin/categories/new" >Criar nova Categoria</a>
15. NO ARQUIVO CategoriesController
16. Criar uma rota para apagar uma categoria específica
17. //rota para realizar a exclusão de uma categoria
18. router.post('/categories/delete',(req, res)=>{
19. var id = req.body.id;
20. if((id != undefined) || (!isNaN(id))){
21. //deletando a categoria
22. Category.destroy({
23. where: {
24. id: id
25. }
26. }).then(()=>{
27. res.redirect('/admin/categories');
28. });
29. }else{
30. res.redirect('/admin/categories');
31. }
32. });

NO ARQUIVO views/categories/index.ejs

1. Configure um form para ao clicar um botão ir para a rota /categories/dele enviando o valor do id da categoria.
2. Substitura o último <th> do forEach.
3. <th>
4. <button class="btn btn-warning">Editar</button>
5. <form  method="POST" action="/categories/delete" style="display: inline;">
6. <input type="hidden" name="id" value="<%=category.id%>">
7. <button class="btn btn-danger">Excluir</button>
8. </form>
9. </th>
10. No final do arquivo após o include do footer. Criar a função
11. Digite :
12. !--criando um script para confirmar a exclusão da categoria-->
13. <script>
14. function confirmarDelecao(event, form){
15. event.preventDefault();
16. //verifica se o usuário quer deletar
17. var decision = confirm('Você quer deletar a categoria?');
18. if(decision){
19. form.submit();
20. };
21. };
22. </script>

NO ARQUIVO CategoriesController.js

1. Criar a rota para abrir o formulário de editar categorias
2. Digite
3. //rota para abrir o formulário para editar a categoria
4. router.get('/admin/categories/edit/:id',(req, res)=>{
5. var id = req.params.id;
6. //será realizado uma pesquisa pelo PK
7. Category.findOne({Where:{id:id}).then(categoria=>{
8. if(categoria != undefined){
9. res.render('admin/categories/edit',{categoria: categoria});
10. }else{
11. res.redirect('/admin/categories');
12. };
13. }).catch(erro=>{
14. res.redirect('/admin/categories');
15. });
16. });

NA PASTA views/categories

1. Criar o arquivo edit.ejs que será a página para edição das categorias

No ARQUIVO edit.ejs

1. Digite
2. <%- include ('../../partials/header.ejs'); %>
3. <%- include ('../../partials/navbar.ejs'); %>
4. <div class="container">
5. <br>
6. <div class="card">
7. <div class="card-header">
8. <h2>Edição de categorias</h2>
9. </div>
10. <div class="card-body">
11. <form method="POST" action="/categories/update">
12. <input type="hidden" name="id" value="<%=category.id %>">
13. <input class="form-control" type="text" name="title" placeholder="Defina o título da categoria" value="<%=category.title%>">
14. <br>
15. <button class="btn btn-primary">Atualizar</button>
16. </form>
17. </div>
18. </div>
19. </div>
20. <%- include ('../../partials/footer.ejs'); %>

NO ARQUIVO categoriesController

1. Crie uma rota para receber os dados do formulário de edição e salvar no banco de dados.
2. Digite
3. /rota para salvar a alteração da categoria
4. router.post('/categories/update',(req, res)=>{
5. var title = req.body.title;
6. var id = req.body.id;
7. var slug = slugify(title);
8. //alterando o valor do campo
9. Category.update({
10. title: title,
11. slug: slug
12. },{where:{
13. id:id
14. }
15. }).then(()=>{
16. res.redirect('/admin/categories');
17. });
18. });

NA PASTA views/admin

1. Crie a pasta articles

NA PASTA views/admin/articles

1. Crie o arquivo new.ejs para cadastrar os artigos

NO ARQUIVO articles/new.ejs

1. Cole o conteúdo do arquivo categories/new.ejs e faça as alterações para mostrar os campos que devem ser preenchidos em categories
2. Digite
3. <%- include ('../../partials/header.ejs'); %>
4. <%- include ('../../partials/navbar.ejs'); %>
5. <div class="container">
6. <br>
7. <div class="card">
8. <div class="card-header">
9. <h2>Novo Artigo</h2>
10. </div>
11. <div class="card-body">
12. <form method="POST" action="/articles/save">
13. <label for="tile">Título:</label>
14. <input class="form-control" type="text" name="title" placeholder="Defina o título do artigo">
15. <label for="cars">Categoria:</label>
16. <select class="form-control" id="cars" name="cars" >
17. <% categories.forEach(category=>{ %>
18. <option value="category.title"><%=category.title%></option>
19. <%})%>
20. </select>
21. <label for="body">Conteúdo:</label>
22. <textarea class="form-control" type="text" name="body" placeholder="Digite o conteúdo do artigo" ></textarea>
24. <br>
25. <button class="btn btn-primary">Cadastrar</button>
26. </form>
27. </div>
28. </div>
29. </div>
30. <%- include ('../../partials/footer.ejs'); %>
31. NO ARQUIVO articles/ArticlesControllers
32. importe o slugify digite ‘const slug = require(‘slugijy’);’
33. importe os models de Category e Articles
34. //importando o model de categorias e artigos
35. const Category = require('./../categories/Category');
36. const Article = require('./Article');
37. Crie a rota para abrir a página de cadastro de categorias
38. Digite
39. //criando a rota de categorias e enviando as categorias para a página new.ejs
40. router.get('/admin/articles',(req,res)=>{
41. Category.findAll({
42. raw:true,
43. order:[['id','DESC']]}).then(categories=>{
44. res.render('admin/articles/new',{categories:categories});
45. });
46. });
47. BAIXANDO E CONFIGURANDO O TinyMCE
48. O tinyMCE é um editor de texto opensource que será configurado na página de artigos dando ‘super-poderes’ para o textarea. Para o usuário configurar todo o artigo antes de salvar.
49. Entre na página [www.tiny.cloud/get-tiny/self-hosted/](http://www.tiny.cloud/get-tiny/self-hosted/)
50. Clicar para baixar no botão TinyMCE Community
51. Abrir a pasta public do projeto.
52. Abrir o arquivo baixado.zip do tiny entrar na pasta ‘/js’ copiar a pasta tinymce e colar dentro da pasta public

NO ARQUIVO view/admin/articles/new.ejs

1. No final da página, importar o tinymce digitando
2. <script src="/tinymce/tinymce.min.js"></script>
3. No <textarea> incluir uma id=’article’. Essa id será fornecida ao tinymce para ele fornecer os recursos extras ao textarea.
4. Após importar, realizar a configuração do tinymce
5. <script>
6. tinymce.init({
7. selector: '#article'
8. })
9. </script>
10. Para melhorar o tinymce instalando mais plugins basta inserir na configuração do tiny conforme abaixo.
11. <script>
12. tinymce.init({
13. selector: '#article',
14. plugins: ['advlist autolink link image lists print preview hr searchreplace wordcount fullscreen insertdatetime media save table paste emoticons']
15. })
16. </script>

TRADUZINDO O TinnyMCE

1. Entrar no endereço
2. <https://www.tiny.cloud/get-tiny/language-packages/>, localizar o indioma português-brasil e clicar em donwload.
3. Abrir o arquivo zip, localizar a pasta ‘langs’ e copiar o arquivo pt\_BR.js para dentro da pasta ‘langs’ que está dentro da pasta public/tinnymce.

NO ARQUIVO view/admin/articles/new.ejs

1. Realizar alteração na configuração do tynni inclua a linguagem.
2. <script>
3. tinymce.init({
4. Language: ‘pt\_BR’,
5. selector: '#article',
6. plugins: ['advlist autolink link image lists print preview hr searchreplace wordcount fullscreen insertdatetime media save table paste emoticons']
7. })
8. </script>
9. Realizado alteração no formulário, para incluir o id da categoria dentro da tag <option> e poder utilizar quando salvar no banco de dados
10. Substitua o texto abaixo no formulário
11. <select class="form-control" id="category" name="category" onselect="">
12. <% categories.forEach(category=>{ %>
13. <option value="<%=category.id%>"><%=category.title%></option>
14. <%})%>
15. </select>

Alterando a página articles/new.ejs, incluído no value da tag <option> a variável id da categoria <option value=”<%=categoy.id%>”>, para utilizar no banco de dados.