Modulo 05

Projeto #01 Criando uma plataforma de perguntas e respostas com Mysql

* Aula 02 versao Bootstrap para esse projeto

<https://getbootstrap.com/docs/4.5/getting-started/introduction/>

* Aula 03 EJS e criação de projeto

Criamos uma nova pasta chamada guiaPerguntas

Vamos iniciar um novo projeto node nessa pasta pelo cmd

npm init -y

vamos instalar o express e o ejs(engine template):

npm install express --save

npm install ejs --save

* Aula 04 Configurando e exibindo HTML com EJS

No arquivo index.js coloque o código

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

crie uma pasta views em seu projeto, onde ficará todos os seus arquivos html

como estamos ultilizando o EJS, na estensão iremos usar o .ejs

crie um arquivo index.ejs na pasta views para ser nossa pagina principal

agora no arquivo index.js

iremos setar o caminho ‘/’, como sendo res.render e não res.send, dizendo o nome do arquivo que está na pasta views, nesse caso o arquivo index.ejs

index.js

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

    res.render('index')

})

Ao utilizar o render ele já vai direto na pasta views, e como se estivesse passando

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

    res.render('views/index')

})

Todo arquivo html deverá ficar na pasta views de seu projeto

* Aula 05 Exibindo variáveis no HTML

Com o EJS e possível passar variáveis do javascript para dentro do html

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

    let nome = 'Rodrigo'

    let lang = 'JavaScript'

    res.render('index')

})

Agora para passar esses valores vamos utilizar essa variáveis dentro do res.render()

index.js

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

    let nome = 'Rodrigo'

    let lang = 'JavaScript'

    res.render('index', {

        nome: nome,

        lang: lang,

        empresa: 'Guia do programador',

        inscrito: 8000

    })

})

Passamos para a view index 4 variaveis(nome, lang, empresa e inscrito)

Agora vamos pegar essa variáveis no arquivo index.ejs, usando a tag

<%= %>, passando o nome da variável

index.ejs

<body>

    <%= nome %>

    <%= lang %>

    <%= empresa %>

    <%= inscrito %>

</body>

Outra coisa que pode fazer e misturar essas tag com html

index.ejs

<body>

    <p>Nome: <%= nome %> </p>

    <p>Linguagem favorita: <%= lang %> </p>

    <p>Empresa: <%= lang %> </p>

    <p>Incritos: <%= inscrito %> </p>

</body>

Passagem de parâmetros obrigatorios direto no url

index.js

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

    let nome = req.params.nome

    let lang = req.params.lang

    res.render('index', {

        nome: nome,

        lang: lang,

    })

})

Os parâmetros obrigatórios passados ficarão salvo em variáveis, que serão chamado no view index

Passagem de parâmetro opcinal

index.js

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

    let nome = req.params.nome

    let lang = req.params.lang

    res.render('index', {

        nome: nome,

        lang: lang,

    })

})

Aqui temos dois parâmetros opacionais nome e lang, que podem ou não ser passado no endereço.

Se no endereço por passado paramentro ficara salvo na variável, se não for passado nada, nada ira ser mostrado

* Aula 06 Estrutura condicionais com EJS

<%= %> -> para chamar variaveis

<% if(){ %> -> para usar expressões

<% } %>

Ou

<% if(){ %> -> para usar expressões

<% } else { %>

<% } %>

index.js

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

    let exibirMsg = true

    res.render('index', {

        msg: exibirMsg

    })

})

index.ejs

<body>

    <% if(msg == true){ %>

        <h3>Isso é uma msg de ERRO!</h3>

    <% }else { %>

        <h3>Nenhum erro!</h3>

    <% } %>

</body>

Nesse caso como a variável msg esta recebendo exibirMsg que por sua vez e true, será mostrada a msg de erro, de dentro do bloco

* Aula 07 Estrutura de repetição com EJS

index.js

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

let produtos = [

        {nome: 'Doritos', preco: 3.14},

        {nome: 'Coca-Cola', preco: 5},

        {nome: 'Leite', preco: 1.45}

    ]

    res.render('index', {

        produtos: produtos

    })

})

No arquivo index.js, estamos enviando como uma variável produtos que está recendo uma lista de produtos, que no nosso arquivo de view poderemos usala chamando um forEach para ler todo o conteúdo dessa lista

index.ejs

<body>

<%= produtos[0].nome %>   <!-- Doritos -->

    <%= produtos[0].preco %>  <!-- 3.14 -->

    <%= produtos[2].nome %>   <!-- Leite -->

    <%= produtos[2].preco %>  <!-- 1.45 -->

</body>

Em nosso arquivo de views podemos recuperar esses valores que estão contido na variável produtos, podemos ultilizamos como desejarmos esses valores

Agora para percorrer essa variável precisamos utilizar um forEach como veremos no código abaixo:

<body>

    <% produtos.forEach(produto => { %> ABERTURA

       <p> <%= produto.nome %> </p>

       <p> <%= produto.preco %> </p>

    <% }) %>   FECHAMENTO

</body>

Observe que ao chamar a função de forEach a abertura dela está em uma tag ejs e seu fechamento em outra tag ejs, e como não e uma variável que estamos chamando e sim uma função a tag ejs não vem acompanhada do ‘=’, já o conteúdo que estamos pegando do forEach esse sim e uma variável por isso estamos chamando com o uso do ‘=’

index.ejs

<body>

    <table>

        <th>Produtos</th>

        <th>Preços</th>

        <% produtos.forEach(produto => { %>

            <tr>

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

                <td><%= produto.preco %> </td>

            </tr>

        <% }) %>

    </table>

</body>

Podemos então montar nossa tabela da maneira que quisermos, ultilizando as tag html e recuperando o dados que foram passados.

* Aula 08 Arquivos estáticos

São arquivos que não são processados no seu backend, como arquivo css, imagens, Javascript do html.

Em nosso projetos criaremos uma pasta chamada public na raiz do nosso projetos, onde ficara todos esses arquivos estáticos, e em sequida informaremos ao express que ultilaremos essa pasta como arquivo estáticos informando o código abaixo:

index.js

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

por padrão no mercado, os arquivos estáticos ficam na pasta public

dentro dessa pasta public iremos criar uma pasta css, e dentro dela criaremos um arquivo style.css

e para utilizar essa folha de estilo iremos em nosso arquivo que desejamos utilizar esse css e fazer o link, como se fosse um html normal

index.ejs

<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>Principal</title>

    <link rel="stylesheet" href="/css/style.css">

</head>

A única direfença que no caminho para linkar o arquivo não e necessário colocar o caminho public, porem e necessário colocar a ‘/’

Para iniciar o endereço do arquivo

<link rel="stylesheet" href="../public/css/style.css">

Ficando assim:

<link rel="stylesheet" href="/css/style.css">

Mesma coisa seria para adicionar uma imagem, criaria uma pasta img dentro de public, colocaria uma imagem dentro dessa pasta e na para chamar esse arquivo usaria a tag img no arquivo de view

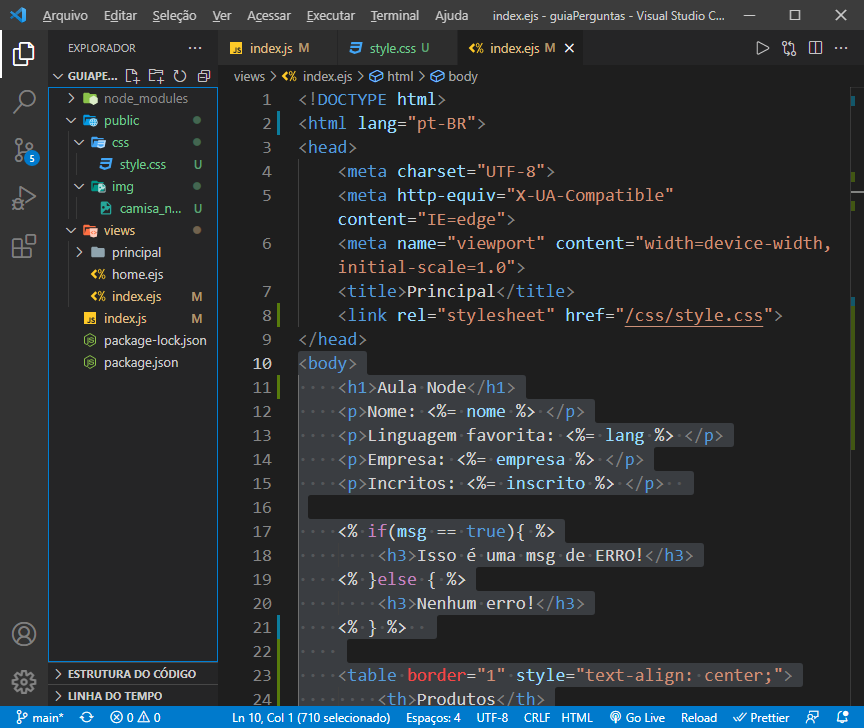
index.ejs

<body>

  <img src="/img/camisa\_node.jpg" alt="">

</body>

Seguindo o mesmo esquema de inserir imagem do html, so mudando o caminho, como esse arquivo está na pasta public, não e necessário informar esse caminho completo, somento o caminho apartir da pasta public começando por ‘/’.



Nosso projeto estará dessa maneira, pasta public para arquivo estáticos(css/img/js), pasta views(index.ejs/sobre.ejs) para as paginas ejs e nosso arquivo index.js sendo nosso arquivo principal onde estão os caminhos completos.

* Aula 09 Apresentando e instalando o Bootstrap

Para criar seu projeto com bootstrap, baixe os arquivos dele e coloque as pastas css e js(baixadas do boot), em seu projeto na pasta public, em sequida faça o link bootstrap.min dos arquivos tanto do css quando do js em seus arquivos de views que iram utilizar o boot

Importante e necessário copiar o link do jquery e do popper e colar em seu projeto também antes do link js do bootstrap

Terá que ficar na ordem, primeiro o jquery, depois popper e por ultimo do bootstrap.min.js, nessa ordem

Index.ejs

<head>

    ...

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

</head>

<body>

    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384- DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkf j" crossorigin="anonymous"></script>

    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/pop per.min.js" integrity="sha384- 9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7l N" crossorigin="anonymous"></script>

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

</body>

Feito isso seu bootstrap estará instalado na sua pagina do projeto

* Aula 10 Criando formulário de perguntas

Ensinamento bootstrap btn

* Aula 11 Adicionando card no formulário

Ensinamento bootstrap card

* Aula 12 Css customizado com bootstrap

Para se trabalhar com css junto com bootstrap, a chamada do link css deve vim depois da chamada do link do bootstrap, para surtir efeito

index.ejs

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

    <link rel="stylesheet" href="/css/style.css">

* Aula 13 Finalizando front com navbar

Ensinando bootstrap navbar

* Aula 14 AVISO: nova forma de utilizar partials – Correção de erro

Antes <%- include partials/header.ejs %>

Agora <%- include (‘partials/header.ejs’) %>

* Aula 15 Reutilizando HTML com partials

Recurso do ejs, que são partes de códigos html que você pode reutilizar essas partes em outros html, assim um recurso e alterado, todos as paginas que tenha esse partials será alterado, sem precisar ficar alternado o código manuamente

**A = 38**

**A = 38**

**A = 38**

**Pagina 1**

**Partials Principal**

**Pagina 2**

Ao alterar o partials todas as paginas que tenham esse partials serão alteradas

**A = 57**

**A = 57**

**Pagina 1**

**Partials Principal**

**A = 57**

**Pagina 2**

Para se utilizar partials, e recomendado na pasta de views criar uma pasta chamada partials, para conter cada pedaço das parciais do seu código

Para incluir basta colocar o código ejs com include seguido do caminho do arquivo dentro de aspas e parênteses:

perguntar.ejs

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

<body>

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

    <h1>Teste de partials</h1>

</body>

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

* Aula 16 Envio de formulário para o Node

Para trabalhar com formulário, a primeira coisa e definir com qual método o formulário vai trabalhar, GET ou POST

<form method="GET">...</form>

Ou

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

GET -> quando os dados desse formulário forem enviado, ele será enviado usando o query params, logo não e tanto recomendado usa-lo formulário com GET

Ex: http://localhost:8080/perguntar?nome=Fulano&senha:123456

POST -> os dados não são passado na url e sim dentro da requisição,

Os dados meio que ficam escondidos ao olho do usuário

A segunda coisa que precisamos fazer e definir para qual rota será enviado esses dados

Para isso e preciso criar uma rota em seu node

index.js

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

    res.send('Formulario Recebido')

})

Você utiliza rota do tipo post geralmente para receber dados de formulário

Agora em nosso formulário dizemos para onde ele irá apontar usando o action

perguntar.ejs

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

    <label for="">Título:</label>

    <input class="form-control" type="text" placeholder="Título">

</form>

Ao enviar esse formulário os dados serão enviado para o caminho

http://localhost:8080/salvarpergunta

agora se precisamos recuperar esses dados enviados

* Aula 17 Capturando dados de um formulário

Para pegar os dados enviado pelo express precismos istalar uma biblioteca chamada body parser em seu projeto

npm install body-parser --save

agora precisamos import o body parser em nosso projeto

index.js

const bodyParser = require('body-parser')

depois temos que linkar o body parser no express, porque o body parser e resposavel por traduzir os dados enviados pelo formulário em uma estrtura Javascript que consiga utilizar no backend

index.js

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

depois podemos colocar um comando opcional bem útil, que permite que a gente leia dado de formulário enviado via json, muito utilizado no uso de api

index.js

app.use(bodyParser.json())

feito isso seu projeto já terá suporte ao body parser, importante que seus campos no formulário tenha nome para que eles possam ser identificados no seu backend

perguntar.ejs

<input type="text" name="titulo">

<textarea name="descricao"></textarea>

E para pegar esse dado passado usamos o código abaixo

index.js

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

  let titulo = req.body.titulo

  let descricao = req.body.descricao

  res.send('Formulario Título: '+titulo +' Descrição: '+descricao)

})

Criamos uma variável e pegamos pela requisição feita no body e o nome da tag que queremos pegar

Nesse caso estamos pegando o valor passado no input que tem o name titulo e descricao

* Aula 18 MySQL Workbench

Interface gráfica para o mysql

* Aula 19 Apresentando e instalando o Sequelize

Para se trabalhar com o mysql e outros bancos de dados sql no node, a melhor maneira e utilizar uma biblioteca chamada sequelize

Com o sequelize você consegue fazer qualquer manipulação no seu banco de dados sql

O sequelize faz todo o processo de conexão de manipulação do banco de dado tudo através no node, então através de código Javascript você consegue manipular o seu banco de dado

Vamos instalar em nosso projeto o sequelize:

npm install sequelize --save

além disso para se trabalhar com o sequelize junto com o mysql, você ainda precisa adicionar uma biblioteca extra, a mysql2 em seu projeto:

npm install mysql2 --save

agora nosso projeto conta com o sequelize e o mysql2

* Aula 20 Conexão com banco de dados

Vamos aprender a conectar nosso sequelize a nosso banco de dados mysql, depois de criar um mysql rodando em sua maquina vamos conectar o mysql

Crie uma pasta em seu projeto que apenas ficará coisas relacionadas a banco de dados, podemos chama la de database

Feito isso crie um arquivo dentro dessa pasta chamado database.js,

Esse arquivo e onde ficará a conexao com o sequelize

Para criar a conexao basta importar o modulo, e após isso construir a sua conexao

database.js

const Sequelize = require('sequelize')

const conection = new Sequelize('guiaperguntas', 'root', '123456', {

    host: 'localhost',

    dialect: 'mysql'

})

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

    host: 'localhost',

    dialect: 'mysql'

})

Nome do banco de dados

Usuário

Senha

Servidor onde está rodando, nesse caso no meu computador

A linguagem utilizada no banco

Feito isso sua conexao estará criada, agora você pode exportar essa conexao para ultilizala em outros arquivos usando o module.exports = connection

batabase.js

const Sequelize = require('sequelize')

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

    host: 'localhost',

    dialect: 'mysql'

})

module.exports = connection

para testar essa conexao você pode testa la

no seu arquivo principal do express, nesse caso o index.js, faca a importação desse arquivo de database criado

index.js

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

agora vamos testar nosso mysql, para isso vamos colocar o código:

index.js

connection.authenticate()

.then(() => {console.log('Conexão feita com o banco de dados!');})

.catch((msgErro) => {console.log('Erro ' +msgErro);})

Agora basta rodar seu projeto no terminal cmd para testar

Caso ocorra algum erro veremos a seguir como resolver um dos principais erro

* Aula 21 Corrigindo erro do conexão do sequelize

Vamos abrir o mysql workbench, clique no primeiro ícone no +sql,

E digite o seguinte código, para alterar o modo de altenticação do usuário root do seu servidor mysql para a sua senha atual

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password

BY '123456';

Apenas altere em seu projeto o usuário, local do banco e senha

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password

BY '123456';

Usuário

Local do servidor

Senha

* Aula 22 Gerando tabela com model

Para criar uma tabela usaremos um conceito chamado model, o model representa a sua tabela, e uma estrutura de dados que representa a sua tabela

Vamos criar esse model dentro da pasta database

Iremos criar um arquivo chamado Pergunta.js

Como esse arquivo e um model, uma boa pratica e começar o nome do arquivo com a letra maiúscula, para poder identificar facilmente que isso e um model

Para definir um model temos que importar o sequelize e a conexão com o banco de dado(a conexão esta na mesma pasta basta colocar ‘./nome\_da\_conexao’ que ele já importa)

Pergunta.js

const Sequelize = require('sequelize')

const connection = require('./database')

agora vamos definir nosso model:

Pergunta.js

const Pergunta = connection.define('pergunta', {

    titulo:{

        type: Sequelize.STRING,

        allowNull: false

    },

    descricao:{

        type: Sequelize.TEXT,

        allowNull: fase

    }

}, {})

Nome da tabela

Campos

Tipo do campo

Se o valor pode ser vazio

Isso vai criar uma tabela no seu banco de dados com o campo titulo e descricao

Elem disso precisamos passar um json de opção ao final ou não passar se não houver

Para de fato criar essa tabela no banco iremos fazer:

Pergunta.js

Pergunta.sync({force:false}).then(() => console.log('Tabela criada')

)

O uso do force sendo falso indica que caso essa tabela já exista o código não ira tentar forçar a criar essa tabela pois ela já existe

Agora so precisamos executar isso no nosso arquivo index.js

Primeiro importaremos esse model no index.js

const perguntaModel = require('./database/Pergunta')

só pelo fato de fazer o import em nosso arquivo principal, essa tabela já será criada

* Aula 23 Salvando perguntas no banco de dados

index.js

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

    let titulo = req.body.titulo

    let descricao = req.body.descricao

    res.send('Formulario Recebido: Título: '+titulo +' Descrição: '+descricao )

})

Agora vamos pegar os dados que foram enviado aqui e salvar no banco de dados

Para manipular uma tabela no sequelize precisamos importar o model dessa tabela no arquivo principal

Pergunta.js

module.exports = Pergunta

agora no arquivo index.js vamos importar esse module de Pergunta,

já fizemos esse import antes, porem so iremos alterar o nome dele de perguntaModel para Pergunta:

index.js

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

agora com esse model em mãos, e vamos onde os dados do formulário chegam e colocamos o código:

index.js

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

    let titulo = req.body.titulo

    let descricao = req.body.descricao

    Pergunta.create({

        titulo: titulo,

        descricao: descricao

    })

})

o método create e responsável por salvar uma pergunta no banco de dado no model que está chamando ele, nesse caso o model Pergunta, o primeiro campo titulo se refere a coluna da tabela e o segundo campo titulo se refere ao valor da requicao passado na variável

index.js

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

    let titulo = req.body.titulo

    let descricao = req.body.descricao

    Pergunta.create({

        titulo: titulo,

        descricao: descricao

    }).then(() => {

        res.redirect('/')

    })

})

Aqui estamos usando caso o valor seja passado ao banco de dado, a pagina será redirecionada a pagina principal

Nosso formulário em perguntar.ejs envia um formulário com o método post ao caminho ‘/salvarpergunta’, que recebe duas requisições através do name que viram duas variáveis, após isso o model Pergunta usa a função create que salva essas variáveis em seus respectivos campos informados, e por final se tudo de certo a pagina e redirecionada a pagina principal e os dados enviados ao banco de dados

* Aula 24 Buscando perguntas no banco de dados

Vamos agora listar as perguntas indo ate a nossa rota principal ‘/’

index.js

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

    Pergunta.findAll()

    res.render('index.ejs')

})

Para recuperar os dados informamos de qual model desejamos recuperar esses valores nesse caso o model Pergunta e colocamos o método findAll(), que e o equivalente ao SELECT \* FROM perguntas

Não bastas so fazer isso, precisamos receber essa lista de perguntas com o uso then que joga todos os valores em uma variavel

index.js

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

    Pergunta.findAll({raw: true}).then(perguntas => {

        res.render('index.ejs', {

            perguntas: perguntas

        })

    })

})

Ao acessar esse caminho, a função findAll irá fazer um select no model Pergunta, que ao finalizar a busca irá jogar essa busca em uma variável perguntas que por sua vez irá renderizar a pagina index.ejs passando como paramentro perguntas que receberá o valor de perguntas

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

    Pergunta.findAll() .then( perguntas => {

        res.render('index.ejs', {

            perguntas: perguntas

        })

    })

})

Espécie de select \* from no model Pergunta, essa função ira percorrer o model Pergunta

Ao finalizar essa busca, esse valor da busca ira para perguntas, que iniciará uma função que ira renderizar a pagina index.ejs, essa pagina recebera um parâmetro chamado perguntas que recebera os valores da pesquisa de findAll que gerou perguntas

* Aula 25 Listando perguntas no front end

Agora que já podemos usar o parâmetro em index.ejs, vamos pegalo com o uso do ejs, nosso caminho esta recebendo um parâmetro chamado de perguntas que e um array

index.ejs

<div class="container">

        <% perguntas.forEach(pergunta => { %>

            <p><%= pergunta.titulo %> </p>

            <p><%= pergunta.descricao %> </p>

        <% }) %>

    </div>

Agora podemos recuperar os valores dos campos títulos e descricao de cada linha de nossa tabela

Agora só fazer a estilização que desejar

index.ejs

<div class="container">

        <h1>Perguntas</h1>

<hr>

        <a href="/perguntar" class="btn btn-primary">Perguntar</a>

<hr>

        <% perguntas.forEach(pergunta => { %>

            <div class="card ">

                <div class="card body">

                    <h3><%= pergunta.titulo %> </h3>

                </div>

                <div class="card-footer">

                    <button class="btn btn-primary">

Responder

</button>

                </div>

            </div>

        <% }) %>

    </div>

* Aula 26 Ordenação com Sequelize

Por padrão a ordem de exibição e de acordo que os valores foram inseridos, agora caso eu queira mudar essa ordem terei que colocar o código:

index.js

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

    Pergunta.findAll({raw: true, order: [

        ['id', 'DESC'] // ASC = crescente || DESC = Decrescente

    ]}).then(perguntas => {

        res.render('index.ejs', {

            perguntas: perguntas

        })

    })

})

Na função de findAll() dentro dela eu coloco um objeto order, com uma array de dois valores, no primeiro o campo que quero trabalha, nesse caso o campo ID e no segundo valor a ordem que quero, DESC( decrescente) ou ASC (crescente), fazendo isso a ordem de id que antes ia de 1, 2,3,4 agora fica decrescente 4, 3, 2, 1 do maior para o menor, também pode ser por letra,ASC a, b, c, d ou DESC d, c, b, a

* Aula 27 Busca condicional com Sequelize

Iremos criar rotas para cada pergunta, de acordo com o id delas

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

    let id = req.params.id

    Pergunta.findOne({

        where: {id: id}

    }).then(pergunta => {

        if(pergunta != undefined){

            res.render('pergunta')

        } else {

            res.redirect('/')

        }

    })

})

Criamos um caminho /pergunta/:id, onde pegamos esse parâmetro id do endereço e colocamos numa variável id

Com a função findOne(), ele irá procura apenas uma linha ao contrario do findAll() que percorre todas as linhas, e colocamos uma condição where que verifica a coluna id da tabela que seja parecida com a variável id que recebeu o parâmetro no endereço, assim que esse valor for encontrado essa informação irá para a variável pergunta, que em seguida o código entrará em uma condição if, se a variável retornar algum valor, a resposta será uma pagina, caso esse valor retorne valor underfined a pagina sera redirecionada para a pagina principal

* Aula 28 Front end da página de perguntas

index.js

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

    let id = req.params.id

    Pergunta.findOne({

        where: {id: id}

    }).then(pergunta => {

        if(pergunta != undefined){

            res.render('pergunta', {

                pergunta: pergunta

            })

        } else {

            res.redirect('/')

        }

    })

})

Se a pesquisa retornar algum valor, a res.render mostrara a pagina pergunta de views e receberá um parâmetro pergunta, que recebe esse valor encontrado jogado em pergunta

pergunta.ejs

<body>

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

    <div class="container">

        <hr>

        <h1><%- pergunta.titulo %> </h1>

        <p><%- pergunta.descricao %> </p>

        <hr>

    </div>

</body>

Aqui estamos pegando o parâmetro pergunta que foi passado e usando os campos da tabela que desejamos, nesse caso estamos pegando a coluna titulo e descricao

* Pagina 29 Perguntas com links dinâmicos

index.ejs

 <% perguntas.forEach(pergunta => { %>

    <div class="card ">

      <div class="card body">

        <h3><%= pergunta.titulo %> </h3>

      </div>

      <div class="card-footer">

        <a href="/pergunta/<%= pergunta.id %> ">

          <button class="btn btn-primary">Responder</button>

           </a>

     </div>

     </div>

<% }) %>

Para fazer a pagina ficar dinâmica a pagina, temos um botão em cada pergunta, esse botão redireciona para um link ‘/pergunta/pergunta.id’

Como estamos dentro de uma forEach que esta percorrendo toda a tabela, conseguimos recuperar o id, titulo, descricao e outros valores,

Então com o uso do ejs, conseguimos passar esse valor de cada id na variável do href

Como o caminho completo conseguimos acessar a rota de cada endereço