ROTEIRO PARA CRIAR VALIDAÇÃO DE FORMULÁRIO COM O EXPRESS

1. Caso seja um novo projeto
2. Criea a pasta Validacao
3. Realize a inicialização com : “npm init”
4. Baixe as bibliotecas:
5. “npm install express –save”
6. “npm install body-parser --save”
7. “npm install express-session --save”
8. “npm install ejs –save”
9. “npm install express-flash –save”
10. Crie o arquivo index.js
11. No arquivo index.js
12. Realize a importação das bibliotecas
13. //importando as bibliotecas
14. const express = require("express");
15. const session = require("express-session");
16. const flash = require("express-flash");
17. Instancie o express
18. //instancia do express
19. const app = express();
20. Configure a view engine e o express
21. /configurando a viw engine
22. app.set('view engine','ejs');
23. //configuração do express
24. app.use(express.urlencoded({extended:false}));
25. app.use(express.json());
26. Configure o express-session
27. //configurando o express-session
28. app.use(session({
29. secret: "Keyboard Cat",
30. resave: false,
31. saveUninitialized:true,
32. cookie:{secure:true}
33. }))
34. Configure o express-flash
35. //configuração do express-flash
36. app.use(flash());
37. Crie uma rota de teste para verificar se aplicação está funcionando
38. //teste para verificar se a aplicação está rodando
39. app.get('/',(*req*,*res*)=>{
40. console.log("Aplicação rodando..");
41. *res*.send("Rodando...");
42. })
43. Configure a aplicação para rodar em uma porta
44. //rodando o servidor
45. app.listen(4030, (*req*,*res*)=>{
46. console.log("Servidor rodando");
47. });
48. Na pasta Validacao
49. Rode o servidor com o comando
50. Nodemon index.js
51. Vá no navegador e digite
52. Localhost:4030/
53. Deverá aparecer a mensagem “Rodando....”
54. Crie uma pasta chamada views
55. Na pasta views crie o arquivo index.ejs que será o nosso formulário
56. No arquivo index.ejs digite as tags do formulário
57. <form *method*="POST" *action*="/form">
58. <input *type*="email" *name*="email" *id*="email" *placeholder*="Digite seu email"><br>
59. <input *type*="text" *name*="nome"  *id*="nome" *placeholder*="Digite se nome"><br>
60. <input *type*="number" *name*="pontos" *id*="pontos" *placeholder*="Digite a pontuação"><br>
61. <button *type*="submit">Enviar</button>
62. </form>
63. No arquivo index.js, vamos renderizar o nosso formulário na rota get “/” que criamos anteriormente.
64. //teste para verificar se a aplicação está rodando
65. app.get('/',(*req*,*res*)=>{
66. // console.log("Aplicação rodando..");
67. //res.send("Rodando...");
68. *res*.render("index");
69. })
70. A validação de formulários pode ser realizada de duas maneiras no front end através de comandos html como o require porém não é seguro esse tipo de validação pois o usuário pode entrar no código fonte da página e remover o require do código, desta forma os dados serão enviados vazios.
71. A outra forma de validação é no backend onde através de uma rota post os dados serão validados segundo uma lógica.
72. No arquivo index.js vamos criar a rota post para receber os dados do formulário e fazer as validações de cada campo individualmente, com as regras de cada campo.Foram criadas variáveis de error para receber a mensagem de erro.

. //rota post que receberá os dados do formulário e fará a validação

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

    //pega os dados passados no corpo da requisição e salva nas variáveis email, nome e ponto

    var {email, nome, pontos} = *req*.body;

    //cria as variaveis de erros que serão tratados

    var emailError, nomeError, pontosError;

    //validação do email caso ele não seja definido ou seja vazio

    if(email == undefined || email == ""){

        emailError = "O email não pode ser vazio";

    }

    //validação do campo pontos

    if(pontos == undefined || pontos < 20){

        pontosError = "A pontuação não pode ser menor de 20";

    }

    //validação do campo nome

    if(nome == undefined || nome == ""){

        nomeError = "O nome não pode ser vazio";

    }

    //verifica se ocorreu qualquer erro de validação

    if(emailError!=undefined || pontosError!=undefined || nomeError!=undefined){

*res*.send("Esse form está com erro");

    }else{

*res*.send("Show de bola este Form");

    }

});

1. Instalar a biblioteca cookie-parser com “npm install –save cookie-parser.
2. Após a instalação vamos importar a biblioteca cookie-parser para o projeto.
3. //importando as bibliotecas
4. const express = require("express");
5. const session = require("express-session");
6. const flash = require("express-flash");
7. const cookieParser = require("cookie-parser");
8. Vamos agora ativar o cookie parser, antes da sessão, vamos também alterar o parâmetro cookie da sessão de secure para maxage, setando o tempo máximo de vida cookie para 60000(milissegundos) que é uma hora.
9. //ativando o cookie parser
10. app.use(cookieParser("palavra secreta"));
11. //configurando o express-session
12. app.use(session({
13. secret: "Keyboard Cat",
14. resave: false,
15. saveUninitialized:true,
16. cookie:{ maxAge:60000 }
17. }))
18. Antes de enviarmos os erros vamos verificar se os erros existem caso contrário iremos seta-los
19. //verifica se o erro existe, caso não exista será setado undefined na variável
20. emailError = (emailError == undefined || emailError == 0) ? undefined : emailError;
21. nomeError = (nomeError == undefined || nomeError == 0 )? undefined : nomeError;
22. pontosError = (pontosError == undefined || pontosError == 0)? undefined: pontosError;
23. Para podermos enviar as mensagens de erro da rota post para outra rota, vamos trabalhar com as sessions-flash que são sessões que duram somente uma vez e depois são destruídas.
24. Vamos definir alqumas session-flash dentro das validações criadas anteriormente.
25. /verifica se ocorreu qualquer erro de validação
26. if(emailError!=undefined || pontosError!=undefined || nomeError!=undefined){
28. //criando sessions-flash para enviar o erro para outra rota
29. *res*.flash("emailError", emailError);
30. *res*.flash("pontosError", pontosError);
31. *res*.flash("nomeError",nomeError);
32. Vamos ainda criar sessions-flash para enviar para o formulário os valores que foram passados via post. Isso para que ao realizar o refresh os campos não fiquem em branco e tenhamos que preencher tudo novamente.
33. //criando flash-sessions para retornar o conteúdo dos campos para o formuláro
34. //novamente e não deixar o campo em brando
35. *req*.flash('conteudoEmail', email);
36. *req*.flash('conteudoNome',nome);
37. *req*.flash('conteudoPontos',pontos);
39. *res*.redirect('/');
41. }else{
42. *res*.send("Show de bola este Form");
43. }
44. Agora na rota “/” vamos criar variáveis que irá receber as mensagens de erro e faremos também uma
45. verificação através de uma condição ternária para ver se a mensagem de erro realmente existe, caso contrário
46. será undefined.
47. //cria as variaveis de erros que receberão os valores das sessions-flash criadas na rota post
48. var emailError = *req*.flash("emailError");
49. var nomeError = *req*.flash("nomeError");
50. var pontosError = *req*.flash("pontosError");
51. ;
53. Vamos verificar também se o erro existe e caso contrário vamos configuralos
54. //verifica se o erro existe, caso não exista será setado undefined na variável
55. emailError = (emailError == undefined || emailError == 0) ? undefined : emailError;
56. nomeError = (nomeError == undefined || nomeError == 0 )? undefined : nomeError;
57. pontosError = (pontosError == undefined || pontosError == 0)? undefined : pontosError;
58. Criamos variáveis para receber os valores dos campos do formulário que chegarm na rota post, estes valores seram passados novamente para o formulário para os campos não fircarem vazios
59. //criando variaveis para receber o conteúdo dos campos e retornalos
60. //para o formulário
61. var email = *req*.flash("conteudoEmail");
62. var nome = *req*.flash('conteudoNome');
63. var pontos = *req*.flash('conteudoPontos');
64. Vamos fazer a mês verificação que fizemos com as variáveis de erro para verificar se as variáveis existem e se possuem algum valor antes de passálas
65. //verifica se existe valores passados senão seta
66. email = (email == undefined || email == 0)? undefined: email;
67. nome = (nome == undefined || nome == 0)? undefined: nome;
68. pontos = (pontos == undefined || pontos < 0)? undefined: pontos;
69. Agora basta passar os erros e as variáveis de valores para a view,
70. *res*.render("index",{emailError, pontosError, nomeError, email: email, nome: nome, pontos: pontos});
71. No arquivo index.ejs
72. Abaixo de cada input crie uma variável que irá receber o erro passado pela rota post e dentro de cada input crie uma variável dentro do campo <value> para receber o valor que foi enviado para a rota post e que agora está retornando.
73. <form *method*="POST" *action*="/form">
74. <input *type*="email" *name*="email" *id*="email" *placeholder*="Digite seu email" *value*="*<*%= email %>" ><br>
75. *<*%= emailError %>
76. <input *type*="text" *name*="nome"  *id*="nome" *placeholder*="Digite se nome" *value*="*<*%= nome %>"><br>
77. *<*%= nomeError %>
78. <input *type*="number" *name*="pontos" *id*="pontos" *placeholder*="Digite a pontuação" *value*="*<*%= pontos %>"><br>
79. *<*%= pontosError %>
80. <button *type*="submit">Enviar</button>
81. </form>
82. OBS: Podemos utilizar a biblioteca validator.js do site npm para realizar validações mais sofisticadas, basta verificar na página https://www.npmjs.com/package/validator