

Programa IT Academy – Processo Seletivo – Edição #18

Nome Completo: Pedro Henrique S.R de Moraes

E-mail: pedrinhu.moraes@gmail.com

Etapa 1 – Questões de lógica

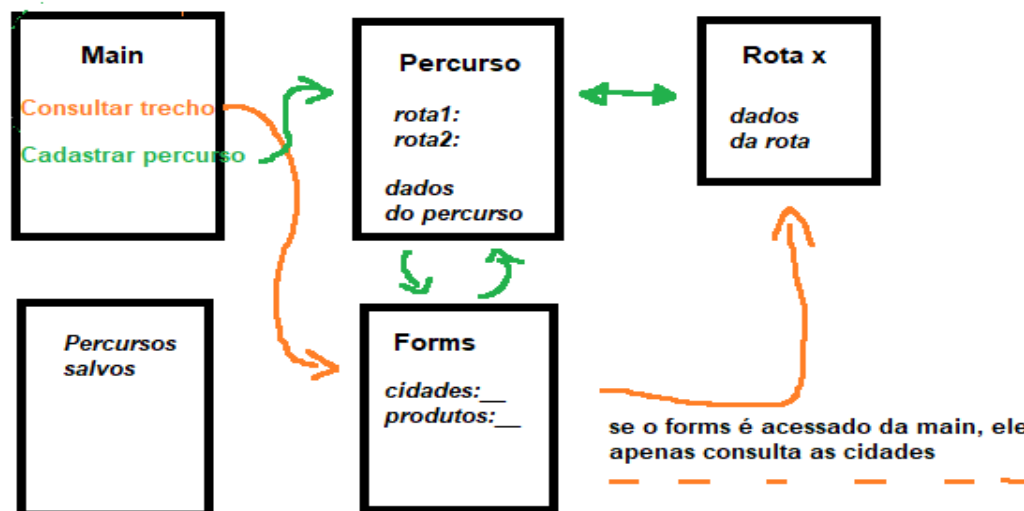
Esta seleção possui 15 questões de lógica de caráter eliminatório. As questões são apresentadas no formulário de Exercício Técnico e devem ser respondidas no próprio formulário online, que deverá ser acessado através do link a seguir: <https://forms.gle/yZtVcv1b5fCgScLBA>

Etapa 2

RESUMO DA SOLUÇÃO

Para esta solução, decidi que iria trabalhar com Javascript e nodejs usando express de framework, com ejs como view engine e, a partir daí, criei o que seria o esboço de como iria funcionar o programa.

A ideia inicial era ter um único formulário, e quando o usuário a partir da página main vai até “criar percurso”, ele pode acessar esse formulário adicionando trecho por trecho do percurso. Assim, se ele quer ir da cidade A até a C passando pela B, ele adiciona o trecho de A até B e depois adiciona o trecho de B até C. O programa então calcula a quantidade de caminhões, distância, custos, etc, de forma independente entre esses locais e na página de PERCURSO fica somente a informação comum a TODO o percurso (como a distância total percorrida de A até C, custo total, etc), enquanto em cada trecho terão as informações específicas àquele trecho (peso transportado de A até B, quantidade de produtos levados de A até B, distância de A até B, etc).



Como visto no print acima do diagrama inicial, a página de percurso ficaria sempre com os dados das rotas salvos, e caso o usuário clique em “salvar”, o percurso com suas respectivas rotas apareceria nos percursos salvos, e caso ele escolhesse só sair e excluir o percurso, os dados seriam perdidos.

Sobre a segunda funcionalidade (Consultar trecho x modalidade) eu preferi utilizar o mesmo formulário, apenas com algumas modificações: caso o usuário venha da página INICIAL (main) o formulário iria omitir os produtos e apenas apresentar a opção “cidade de origem, cidade de destino”, assim levando o usuário a uma página de rotas, porém, essa página de rotas também teria a formatação diferente, pois o usuário poderia apenas consultar o trecho (com informações diferentes) e depois voltar a página inicial.

Outra ideia que tive no decorrer do desenvolvimento que complicou desnecessariamente o projeto mas achei interessante de implementar foi a escalabilidade do projeto. Os dados iniciais de validação (os tipos de produtos, seus pesos, as cidades e as distâncias entre elas) poderiam ser facilmente inseridos dentro do código como constantes (o que inicialmente eu fiz), mas depois pensei “E se a empresa aumentar a quantidade de produtos e adicionar novos pontos de conexões entre cidades?”... Então, decidi trabalhar com dados variáveis de quantidade de tipos de produtos e de cidades. Montei uma tabela no excel com as cidades e produtos e usei a library XLSX do nodejs para pegar os valores das tabelas em arrays de tamanhos indeterminados:

```
//////////pegar os produtos e pesos da tabela
let produto_t = [];
let produto_p = [];

for (let i = 1; i <= 50; i++) {
  if (worksheet[`A${i}`] == null) {
    break;
  }

  let produtos = worksheet[`A${i}`];
  let pesos = worksheet[`B${i}`];

  if (produtos && pesos) {
    produto_t.push(produtos.v);
    produto_p.push(pesos.v);
  }
}
```

(limite de itens: 50)

Assim, durante todo o código eu tive que trabalhar com dois arrays de tipos de produtos (produto_t) e pesos de produtos (produto_p), o que a princípio eu achei que seria o caos, até porque quantos inputs eu forneceria? E como eu iria relacionar o peso ao produto? porém como o index do tipo de produto sempre vai estar relacionado com seu peso (produto_t[x] vai ter o peso do produto_p[x]), e o produto_t[x].length iria dar a quantidade total de produtos, acabou dando bem menos trabalho do que imaginei:

```
<% for (let x=0; x < produto_t.length; x++) { %>
  <label>
    <%= produto_t[x] %>
  </label>
  <input type="number" name="<%= produto_t[x] %>" min="0" max="2000"
    class='produtos' value="<%= parseFloat(produto_q[x]) %>"> <br>

  <% } %>
```

```
////////////////////////////////////TRATA PRODUTO E PESO

let produto_q = [];
let produto_peso = 0;
let peso_total = 0;

for (let i = 0; i < produto_t.length; i++) {

  produto_q[i] = req.body[produto_t[i]]

  produto_peso = produto_q[i] * produto_p[i];
  peso_total += produto_peso;
} ///sessão produto_q , sessão produto_t e sessão peso
```

Os prints acima são dos inputs sendo gerados em ejs e o tratamento desses inputs quando o usuário dá o submit. (produto_q = quantidade de produto). Como deve ter visto nos prints, eu fiz o projeto trabalhando com as express-sessions middlewares do nodejs, o que eu irei detalhar mais à frente.

Parte da minha resolução do desafio, como dito antes, foi a implementação dessa escalabilidade também nas cidades:

```
// CONSTANTES q pegam a primeira row e a primeira coluna da worksheet
const firstRow = XLSX.utils.sheet_to_json(worksheet, { range: 0, header: 1, defval: "" })[0];

const firstColumn = XLSX.utils.sheet_to_json(worksheet, { header: 1, defval: "", raw: false })
  .map(row => row[Object.keys(row)[0]]);

////////////////////////////////FUNCTIONS
function Procura_cidade(procura_row, procura_col) {
  //row
  const row_index = firstRow.findIndex(cellValue => cellValue === procura_row);
  //col
  const col_index = firstColumn.findIndex(cellValue => cellValue === procura_col);

  const cell = { r: row_index, c: col_index };

  return cell;
}

function Distancia_decode(cell) {
  const cell_encode = worksheet[XLSX.utils.encode_cell(cell)].v;
  return cell_encode;
}
```

A ideia foi pegar a primeira coluna e primeira linha da tabela no Excel, analisar os valores e se algum deles correspondesse à cidade, “mesclar” os valores para ter a posição da célula que daria a distância (ex: Cidade1: B1, Cidade2: A3 => B + 3 = B3). A partir daí, eu iria dar encode na célula.value e achar o valor da distância. Aqui estão os prints dessas features funcionando:

Requisição de carga e transporte:

Cidade de origem
Cidade de destino

Celular
Geladeira
Freezer
Cadeira
Luminaria
Lavadora
AAAA
PRODUTOX
PSE

	A	B
1	Celular	0.5
2	Geladeira	60.0
3	Freezer	100.0
4	Cadeira	5.0
5	Luminaria	0.8
6	Lavadora	120.0
7	AAAA	50.0
8	PRODUTOX	1000.0
9	PSE	20.0

[Voltar](#)

(como visto, qualquer alteração na tabela altera os inputs)

Custos

A frota de caminhões mais eficiente para o percurso selecionado foi de:

- Caminhões grandes: 100 , que custaram R\$ 6544440 Reais
- Caminhões médios: 1 , que custaram R\$ 28429.2 Reais
- Caminhões pequenos: 1 , que custaram R\$ 11614.95 Reais
- O custo TOTAL para realizar o transporte até essa rota foi de: R\$ 6584484.15 Reais

Logística:

102 veículo(s) terão que percorrer uma distancia de: 2385 km.

os produtos selecionados a serem transportados de SALVADOR até CURITIBA irão gerar um peso total de: 1004700 kg sendo esses produtos:

Cadeira: 100
Lavadora: 25
AAAA: 20
PRODUTOX: 1000
PSE: 10

[VOLTAR](#)

(e na página de rota, só são mostrados os itens que foram selecionados)

Falando um pouco sobre as sessions, todo o projeto foi montado em cima de 8 sessions de dados, sendo as 7 delas:

```
req.session.peso = [peso_total];  
req.session.caminhao = [n_caminhao];  
req.session.produto_t = [produto_t];  
req.session.produto_q = [produto_q];  
req.session.rota = [rota];  
req.session.preços = [preços_caminhoes_diferentes];  
req.session.distancia = [distancia];
```

Com a oitava sendo a req.session.percurso, que é definida em:

```
let percurso = [req.session.caminhao, req.session.distancia, req.session.preços,  
req.session.produto_q, req.session.produto_t, req.session.rota, req.session.peso]
```

Sendo que a session percurso só é salva (e posteriormente acessada na aba “percursos salvos”) quando o usuário salva ela na aba percurso. As sessions funcionam da seguinte forma:

req.session.percurso[X][Y][Z][W] sendo:

X = index do percurso salvo;

Y = index do dado procurado (0 para n de caminhos, 1 para distancia, 2 para preços...);

Z = index da rota do dado procurado (0 para primeira rota, 1 para segunda, etc);

W = especificação do dado procurado, se existir;

Para exemplificar melhor, vou usar o exemplo do **número de caminhos**:

```
const caminhao_p = [1000, 4.87];  
const caminhao_m = [4000, 11.92];  
const caminhao_g = [10000, 27.44];  
  
//função para achar o número de caminhos P / M / G  
function Achar_n_caminhao(peso_total) {  
  
    let peso = peso_total;  
  
    let n_caminhao_g = 0;  
    let n_caminhao_m = 0;  
    let n_caminhao_p = 0;  
  
    if (peso >= 10000) {  
        n_caminhao_g = Math.floor(peso / caminhao_g[0]);  
        peso = peso % caminhao_g[0];  
    }  
    if (peso > 2000) {  
        n_caminhao_m = Math.floor(peso / caminhao_m[0]);  
        peso = peso % caminhao_m[0];  
        if (peso > 2000) {  
            n_caminhao_m = n_caminhao_m + Math.ceil(peso / caminhao_m[0]);  
            peso = 0;  
        }  
    }  
    if (peso <= 2000) {  
        n_caminhao_p = Math.ceil(peso / caminhao_p[0]);  
    }  
    return [ n_caminhao_g, n_caminhao_m, n_caminhao_p ];  
}
```

Eu desenvolvi a função acima para, um dado peso, ela retornar o número mais eficiente de caminhões pequenos, médios e grandes sendo que a consideração feita é que 3 caminhões pequenos sempre vão ser MENOS eficientes do que um médio e 3 médios MENOS eficientes do que um grande, e a partir disso eu trabalhei com os respectivos pesos de cada caminhão. E quando aplicarmos a função no peso_total (página 2 desse relatório), iremos ter um array com a quantidade de: [caminhões g, caminhões m, caminhões p] que valerá para a rota 1. Quando formos adicionar uma outra rota (de um mesmo percurso) irá acontecer isso:

```
if (!(typeof req.session.rota === "undefined")) {  
  
    req.session.peso.push(peso_total);  
    req.session.caminhao.push(n_caminhao);  
    req.session.produto_t.push([produto_t])  
    req.session.produto_q.push(produto_q)  
    req.session.rota.push(rota);  
    req.session.preços.push(preços_caminhoes_diferentes);  
    req.session.distancia.push(distancia);  
}
```

Se já existir uma req.session.rota contendo uma origem e um destino, as novas sessões serão armazenadas com push(), fazendo com que, se quisermos acessar a quantidade de caminhões pequenos da rota 2, teremos: req.session.caminhao[1][2].

voltando ao percurso, quando o usuário quiser salvar um percurso, acontecerá o seguinte:

```
let percurso = [req.session.caminhao, req.session.distancia, req.session.preços,  
    req.session.produto_q, req.session.produto_t, req.session.rota, req.session.peso]  
  
if (typeof req.session.percurso === "undefined") {  
    req.session.percurso = [percurso];  
} else {  
    req.session.percurso.push(percurso);  
}
```

Se não existir um percurso, ele será salvo como array de [número de caminhões, distancia, preços...], e caso já exista, ele receberá o push com os novos valores de cada session. Isso significa que, se quisermos acessar o valor do número de caminhões pequenos [2], da rota 2[1], do número de caminhões [0], do percurso 3 [2], nós teremos: console.log(req.session.percurso[2][0][1][2]);

Uma questão que me deu certo problema foi a validação. Como eu faço uma análise em uma tabela do Excel, eu não pude fazer a validação das cidades pela parte do cliente, e ao invés disso, criei um bloco try catch:

```
try {  
  req.session.chave = 1;  
  let origem;  
  const destino = req.body.cidade2;  
  
  if (typeof req.body.cidade1 === "undefined") {  
    const i = req.session.rota.length - 1;  
    origem = req.session.rota[i][1];  
  } else {  
    origem = req.body.cidade1;  
  }  
  const cell = Procura_cidade(origem, destino);  
  const distancia = Distancia_decode(cell); //sessão distancia  
  
  const rota = [origem, destino] //sessão rota
```

Quando o usuário enviasse o formulário, o try iria mudar a session_chave para 1 e iria tentar executar as funções de Procura_cidade e Distancia_decode (vistas na página 3), e caso ele não conseguisse o catch iria mudar a req.session.chave para 0 e recarregaria a página:

```
catch {  
  req.session.chave = 0;  
  res.redirect('/req_carga-transporte')  
}
```

```
<% if (chave==0) { %>  
  <script> alert("Cidade(s) não encontrada(s), verifique o manual e tente novamente.") </script>  
  <% } %>  
  
<% if (typeof nova_origem === "undefined" ) { %>
```

e como a página seria carregada com a chave = 0, rodaria o script de cidades não encontradas. Após o envio do formulário, a chave volta a ser 1, para que não apareça a mensagem de erro no futuro caso o formulário seja enviado corretamente pelo try.

Outro processo de validação que fiz foram com as quantidades de produtos, como a aba “Salvar percurso” exige o transporte de ao menos uma unidade de um produto, eu utilizei um event listener para que a quantidade de algum dos produtos tenha que ser diferente de 0, e aproveitando isso, também criei uma validação do lado do cliente para que as abas de cidade1 e cidade2 não possam estar em branco:

```
public > public.js > JS public.js > addEventListener('submit', callback
1  const produtos = document.querySelectorAll('.produtos');
2  const cidades = document.querySelectorAll('.cidade');
3
4  let prod_state = false;
5
6  addEventListener('submit', (e) => {
7      if ( cidades[0].value == "" || cidades[1].value == ""){
8          alert("Preencha o campo Cidade de origem e Cidade de destino")
9          e.preventDefault();
10     }
11
12
13     for (let i = 0; i < produtos.length; i++){
14
15         if (produtos[i].value != 0 && produtos[i].value != ""){
16             prod_state = true;
17             break;
18         }
19     }
20     if (prod_state == false){
21         alert("Selecione pelo menos um item a ser transportado")
22         e.preventDefault();
23     }
24 });
```

Também é válido comentar sobre como o forms de requisição de carga e transporte (pág 4) se comporta quando é chamado da página main e da página de criação de percurso. Na página main uma variável de sessão user é definida como 0, e na página de percurso é definida como 1, e quando o usuário sai da página de percurso para a main TODAS as req.sessions são deletadas, EXCETO a req.session.percurso que NUNCA é deletada.

```
// GET E POST
const pm_get = (req, res) => {

    if (!(typeof req.session.rota === "undefined")){

        delete req.session.peso;
        delete req.session.caminhao;
        delete req.session.produto_t;
        delete req.session.produto_q;
        delete req.session.rota;
        delete req.session.preços;
        delete req.session.distancia;

    }

    req.session.user = 0;

    res.render('page-main.ejs');
```


Isso faz com que da próxima vez que ele vá para página de percurso, a página esteja em branco porque as sessões que guardam os dados não existem e a req.session.percurso só é exibida na página de dados. E, como dito antes, o user =0 fará com que só parte do conteúdo da página forms seja exibida, e o mesmo acontecerá com a página de resultado(rota), que terá uma formatação diferente caso o usuário venha direto da pagina inicial:

```
<% //////////////////////////////////////////////////CASO SEJA SOLICITAÇÃO SIMPLES
if (user==0){ %>

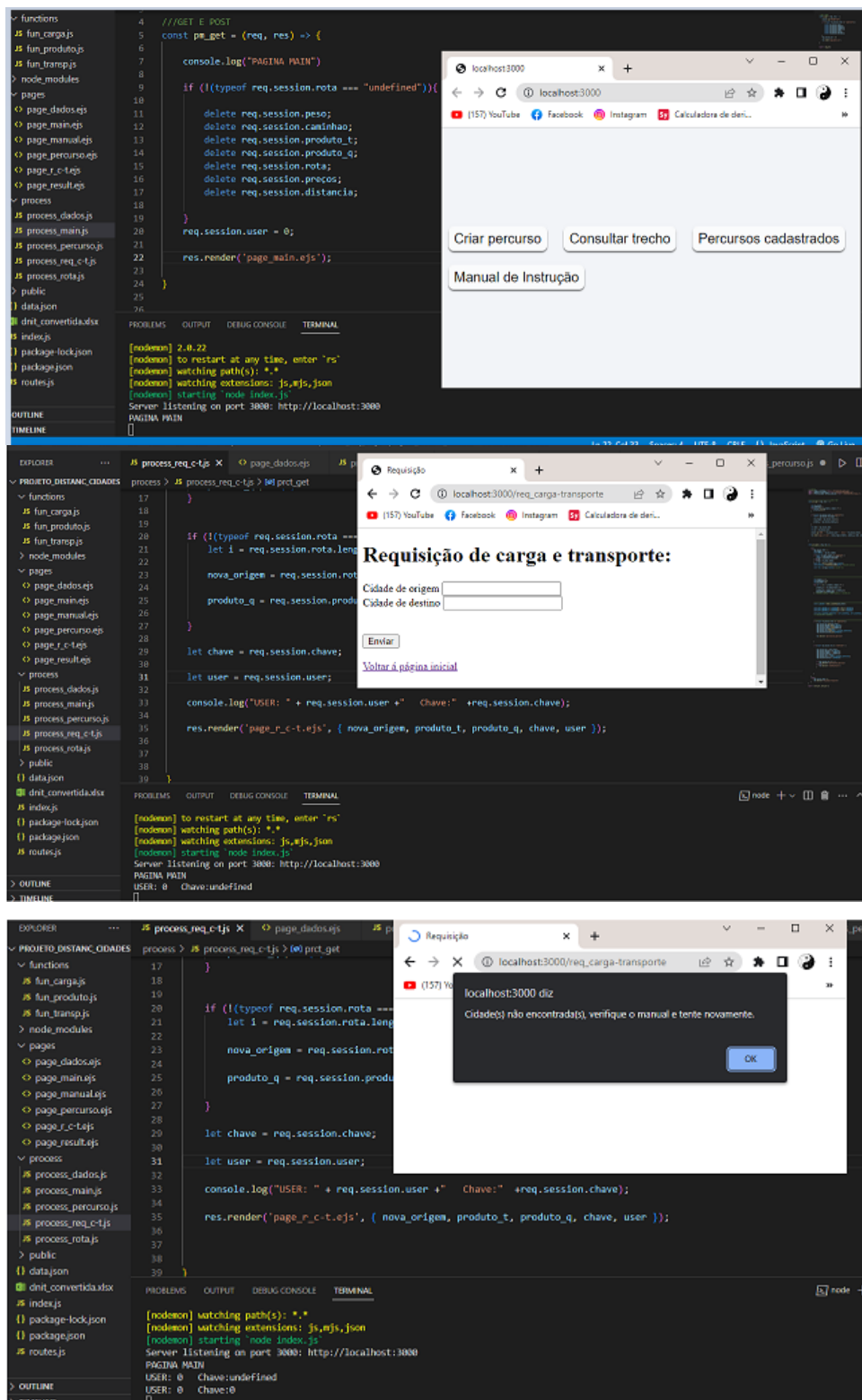
    <h2>A distancia de <%= rotas[0] %> até <%= rotas[1] %> é de: <%= distancia %> km
    </h2> <br>
    <label> Os respectivos caminhões e custos para esse percurso seriam dê: </label>
    <br>
    <ul>
        <li> Caminhão pequeno (uma tonelada): custo total seria de R$ <%= (distancia * 4.87)
            %> reais</li>
        <li> Caminhão médio (quatro toneladas): custo total seria de R$ <%= (distancia *
            11.92) %> reais</li>
        <li> Caminhão grande (dez toneladas): custo total seria de R$ <%= (distancia *
            27.44) %> reais</li>
    </ul>

    <a href="/"> Sair</a>

    <% } %>
```

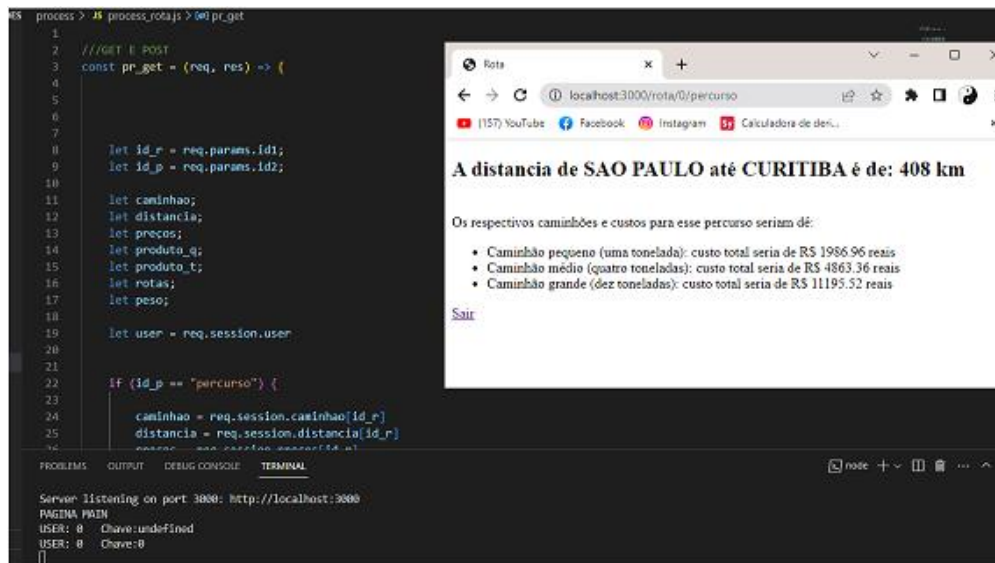
TESTES (aqui você deverá colar capturas de tela de todas as funcionalidades desenvolvidas e realizar comentários, use o espaço que julgar necessário)

FUNCIONALIDADE 1 – Consultar trechos x modalidade



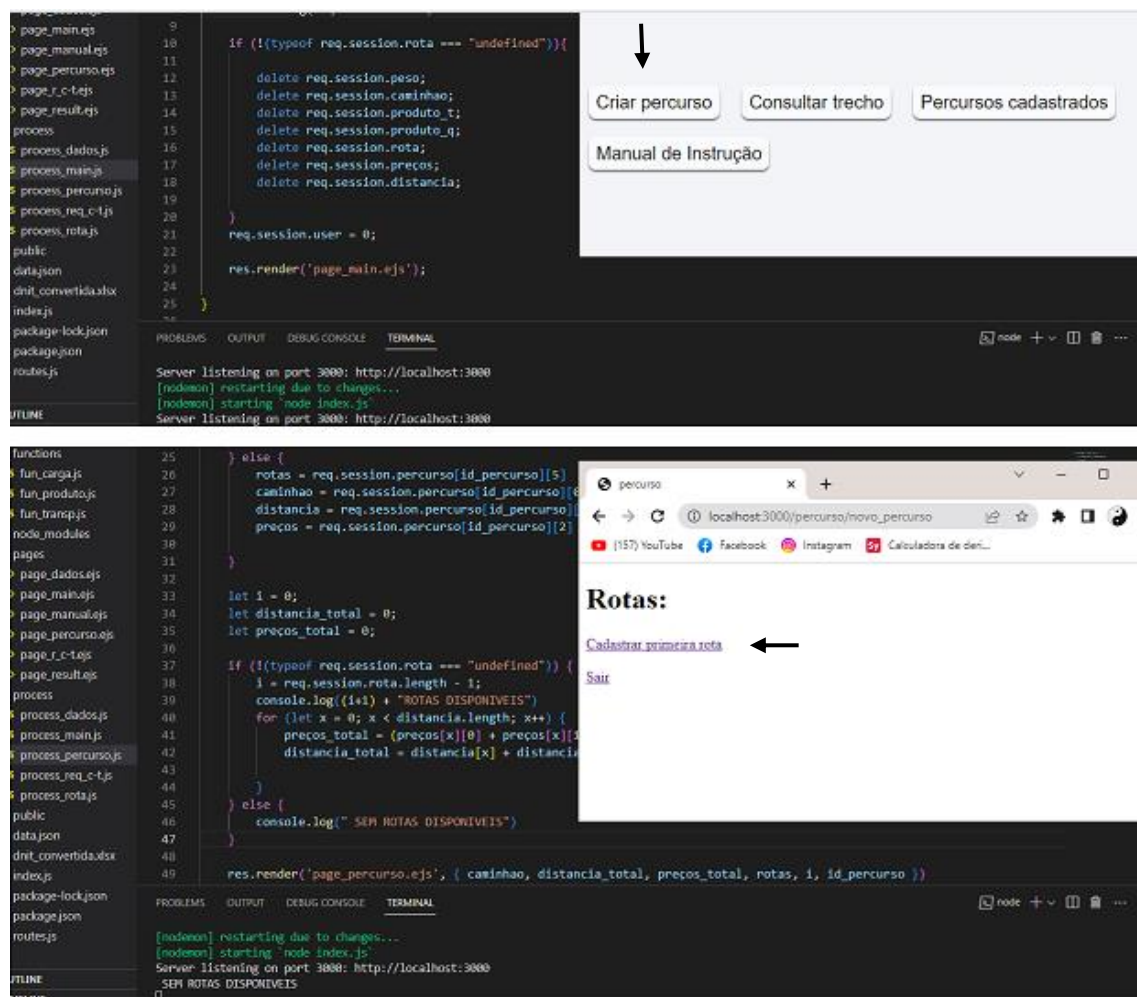
The screenshots illustrate the development and testing of a web application. The first screenshot shows the initial page with buttons for 'Criar percurso', 'Consultar trecho', and 'Percurso cadastrados'. The second screenshot shows the 'Consultar trecho' form with fields for 'Cidade de origem' and 'Cidade de destino'. The third screenshot shows an error message 'Cidade(s) não encontrada(s), verifique o manual e tente novamente.' when invalid destinations are entered.

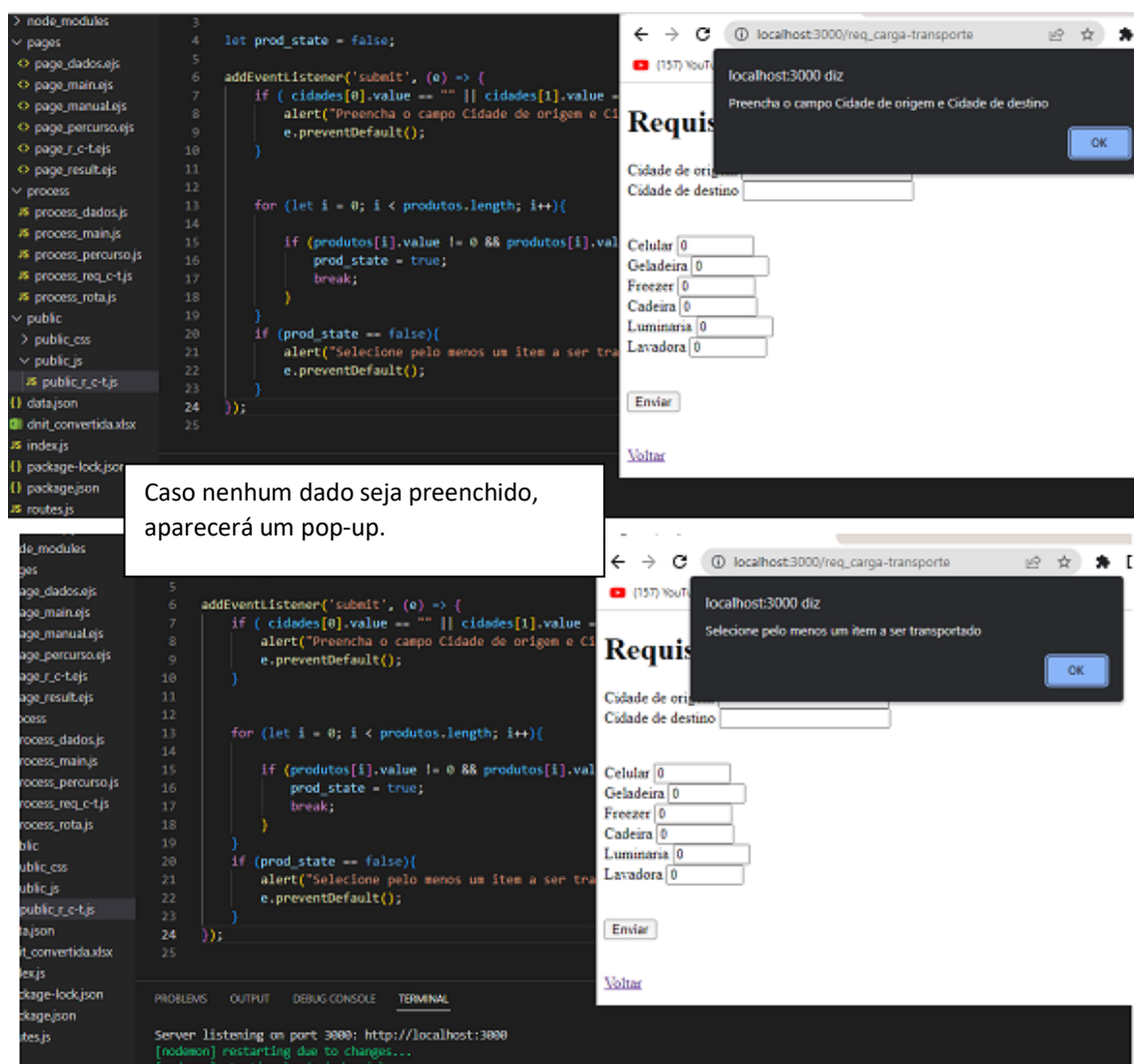
Caso os destinos
sejam inválidos



Caso os destinos sejam corretos, como SAO PAULO e CURITIBA

FUNCIONALIDADE 2 – Cadastrar transporte





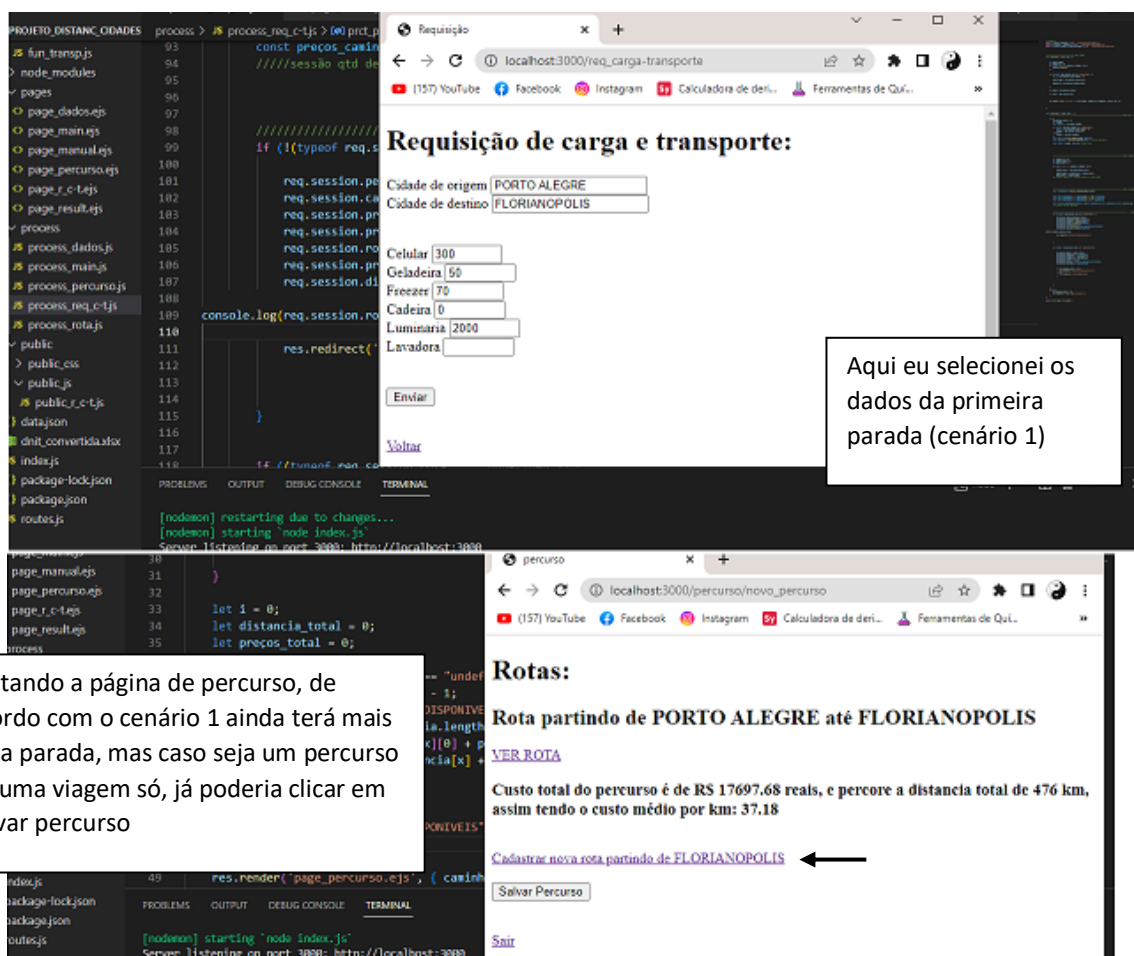
Caso nenhum dado seja preenchido, aparecerá um pop-up.

```

3 let prod_state = false;
4
5
6 addEventListener('submit', (e) => {
7   if ( cidades[0].value == "" || cidades[1].value == "" ) {
8     alert("Preencha o campo Cidade de origem e Cidade de destino");
9     e.preventDefault();
10  }
11
12  for (let i = 0; i < produtos.length; i++){
13
14    if (produtos[i].value != 0 && produtos[i].value != 0) {
15      prod_state = true;
16      break;
17    }
18  }
19
20  if (prod_state == false){
21    alert("Selecione pelo menos um item a ser transportado");
22    e.preventDefault();
23  }
24 });
25

```

Server listening on port 3000: http://localhost:3000
[nodemon] restarting due to changes...
[nodemon] starting 'node index.js'



The image shows a development environment with a code editor on the left and a web browser on the right. The code editor displays JavaScript code for a web application, including file names like `fun_trans.js`, `node_modules`, `pages`, `page_dados.js`, `page_manuais.js`, `page_percurso.js`, `page_r_c_t.js`, `page_result.js`, `process`, `process_dados.js`, `process_main.js`, `process_percurso.js`, `process_req_c_t.js`, `process_rotas.js`, `public`, `public_css`, `public_js`, `public_r_c_t.js`, `datajson`, `dnit_convertidafox`, `index.js`, `package-lock.json`, `package.json`, and `routes.js`. The browser shows two pages: 'Requisição de carga e transporte' and 'Rotas'.

Requisição de carga e transporte:

Cidade de origem: PORTO ALEGRE
 Cidade de destino: FLORIANOPOLIS
 Celular: 300
 Geladeira: 50
 Freezer: 70
 Cadeira: 0
 Luminaria: 2000
 Lavadora:
 Enviar

Aqui eu seleccionei os dados da primeira parada (cenário 1)

Rotas:

Rota partindo de PORTO ALEGRE até FLORIANOPOLIS

[VER ROTA](#)

Custo total do percurso é de R\$ 17697,68 reais, e percorre a distancia total de 476 km, assim tendo o custo médio por km: 37.18

[Cadastrar nova rota partindo de FLORIANOPOLIS](#)

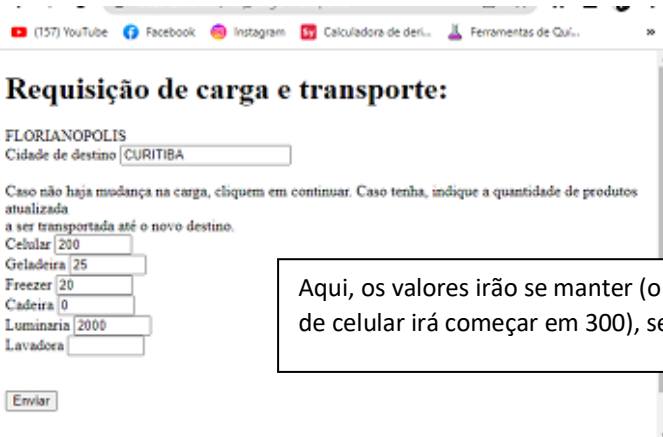
[Salvar Percurso](#)

Voltando a página de percurso, de acordo com o cenário 1 ainda terá mais uma parada, mas caso seja um percurso de uma viagem só, já poderia clicar em salvar percurso


```

28 distancia = req.session.percursos[
29 precos = req.session.percursos[id_p
30
31 }
32
33 let i = 0;
34 let distancia_total = 0;
35 let precos_total = 0;
36
37 if (!typeof req.session.rotas === "und
38 i = req.session.rotas.length - 1;
39 console.log((i+1) + "ROTAS DISPONÍ
40 for (let x = 0; x < distancia.length
41   precos_total = (precos[x][0] +
42     distancia_total = distancia[x]
43   }
44 }
45 } else {
46   console.log("SEM ROTAS DISPONÍVEI
47 }
48
49 res.render('page_percursos.ejs', { cami

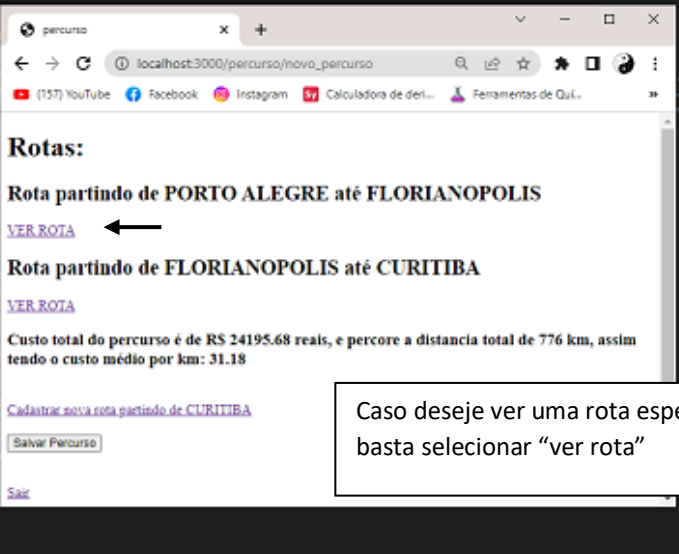
```



```

25 } else {
26   rotas = req.session.percursos[id_p
27   caminho = req.session.percursos[id
28   distancia = req.session.percursos[id
29   precos = req.session.percursos[id_p
30
31 }
32
33 let i = 0;
34 let distancia_total = 0;
35 let precos_total = 0;
36
37 if (!typeof req.session.rotas === "und
38 i = req.session.rotas.length - 1;
39 console.log((i+1) + "ROTAS DISPONÍ
40 for (let x = 0; x < distancia.length
41   precos_total = (precos[x][0] +
42     distancia_total = distancia[x]
43   }
44 }
45 } else {
46   console.log("SEM ROTAS DISPONÍVEI
47 }
48
49 res.render('page_percursos.ejs', { cami

```



```


distancia = req.session.percursos[id_p
precos = req.session.percursos[id_p

let i = 0;
let distancia_total = 0;
let precos_total = 0;

if (!typeof req.session.rotas === "und
i = req.session.rotas.length - 1;
console.log((i+1) + "ROTAS DISPONÍ
for (let x = 0; x < distancia.length
  precos_total = (precos[x][0] +
    distancia_total = distancia[x]
  }
}
} else {
  console.log("SEM ROTAS DISPONÍVEI
}

res.render('page_percursos.ejs', { cami

```



FUNCIONALIDADE 3 – Dados estatísticos

```
3 //GET E POST
4 const pd_get = (req, res) => {
5
6   let percurso = [];
7
8   if (!typeof req.session.percurso === "undefined"){
9     percurso = req.session.percurso
10   }
11
12
13
14   console.log(req.session.percurso[0][2][0][0])
15   res.render('page_dados.ejs', {percurso});
16
17 }
18
19
20
21
22 export { pd_get };
23
```

ssc [[0, 0, 1986.06]]
[[0, 0, 1986.06]]
[[0, 0, 1986.06]]
[nodeemon] restarting due to changes...
[nodeemon] starting "node index.js"

localhost:3000/dados

(157) YouTube Facebook Instagram Calc

Percurso 1:
[VER PERCURSO](#)

Percurso 2:
[VER PERCURSO](#)

Percurso 3:
[VER PERCURSO](#)
[Voltar](#)

Em “salvar percurso” é possível ver os percursos até em tão salvos, e clicando em determinado percurso, é possível ver dados gerais desse percurso e dados específicos de cada rota dele. Igual a tela de criação de percurso, porém sem a opção de inserir novas rotas.

Rotas:

Rota partindo de SALVADOR até ARACAJU
[VER ROTA](#)

Rota partindo de ARACAJU até PORTO ALEGRE
[VER ROTA](#)

Custo total do percurso é de R\$ 17785.24 reais, e percorre a distancia total de 3652 km, assim tendo o custo médio por km: 4.87

[Voltar](#)
[Sair](#)

AUTOAVALIAÇÃO

Você concluiu a implementação de 100% das funcionalidades solicitadas?

(x) Sim () Não

Para as 3 principais funcionalidades solicitadas, como você avalia a sua solução?

Marque um 'X'.

	Inexistente/ Insuficiente	Pouco satisfeito(a)	Satisfeito(a)	Muito satisfeito(a)
Funcionalidade 1				x
Funcionalidade 2			x	
Funcionalidade 3			x	

Principais dificuldades

Acho que a principal dificuldade foi me organizar para criar algo intuitivo para alguém que não criou o projeto entender o que eu estava fazendo ou conseguir utilizar de maneira eficiente. Por mais que eu tenha criado um manual (que está no projeto e também na pasta zipada), sinto que algumas coisas podem ter ficado “confusas”, e imagino que quando forem ler meu código no arquivo zipado irão ver que eu tentei reaproveitar muito o código, a um ponto que algumas variáveis que parecem inúteis porque estão soltas na verdade tão desempenhando um papel super importante.

Outra dificuldade foi em relação ao rearranjo das sessões, eu estou habituado a salvar e resgatar em um banco de dados qualquer informação relevante, e trabalhar com quase todos os dados armazenados em memória foi um desafio.

Desempenho Geral

Eu fiquei parcialmente feliz com meu resultado, dentro do meu programa eu consigo realizar todos os cenários propostos solicitados (cenário 1, cenário 2, solicitação de trecho, mostrar dados referentes ao percurso e trecho) porém senti que eles ficaram expostos de uma forma meio confusa e que talvez NodeJs não seria a melhor ferramenta para o projeto, já que um desafio assim praticamente pede por um programa inteiramente desktop, e a tentativa de criar um site que realiza a função ficou meio amarga por causa de algumas limitações do projeto (como não poder usar banco de dados) e algumas limitações próprias (inexperiência com frameworks de frontend, por exemplo).

Pelos motivos já descritos, eu fiquei satisfeito com as funcionalidades 2 e 3, pois elas cumprem os cenários solicitados, porém, elas poderiam ter sido melhores executadas se eu tivesse tomado um outro caminho para o problema no início do projeto.

Obrigado por participar deste processo seletivo.
Salve o documento em PDF com o seu nome completo.