

Node JS

Exercício extras

Unidade 1

1. Dada a função *arrow*, declarada na linha 1 do programa abaixo, qual será a declaração equivalente dela usando o modelo regular?

Figura 01 – Função *arrow* exercício

```
JS src5.js ●
1 let minhaFunc = (param1, param2) => param1 + param2;
2 console.log(`Minha função retorna: ${minhaFunc(2, 2)}`);
3
```

Fonte: Elaborada pelo autor (2021).

Solução:

```
JS src6.js x
1 function minhaFunc (param1, param2) {
2   |   return param1 + param2;
3 }
4
5 console.log(`Minha função retorna: ${minhaFunc(2, 2)}`);
6
```

Fonte: Elaborada pelo autor (2021).

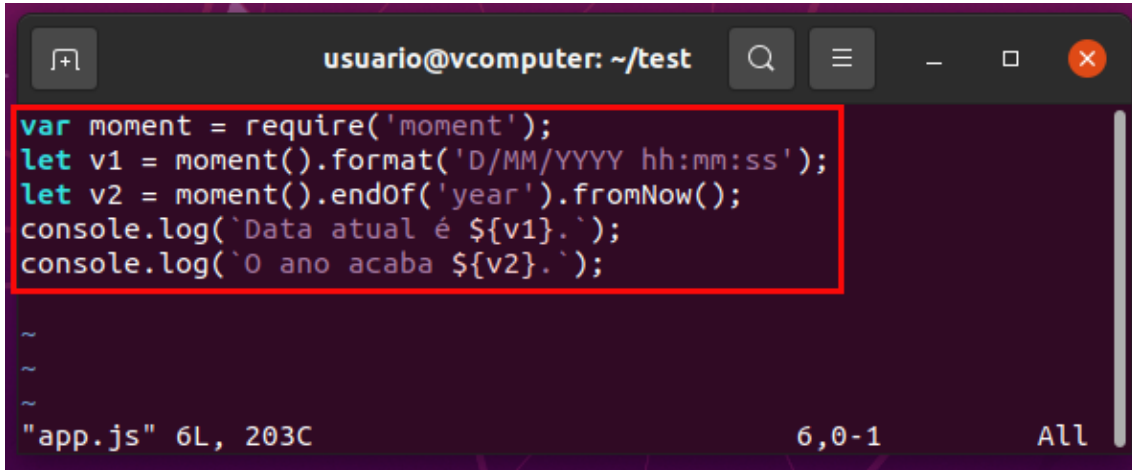
2 Escreva um programa para o Node.js que retorne a seguinte saída: ele precisa retornar a data/hora atual e também mostrar, no final do programa, o tempo restante até o fim do ano corrente.

Figura 03 – A saída desejada para o segundo exercício

Solução:

- A resolução pode variar, mas recomenda-se a utilização da lib 'Moment', pois ela é capaz de realizar a contabilidade relativa com datas, algo que é exigido neste exercício. Uma referência será deixada abaixo para servir de guia:

Figura 04 – A saída desejada para o segundo



```
usuario@vcomputer: ~/test
var moment = require('moment');
let v1 = moment().format('D/MM/YYYY hh:mm:ss');
let v2 = moment().endOf('year').fromNow();
console.log(`Data atual é ${v1}.`);
console.log(`O ano acaba ${v2}.`);

~
~
~
"app.js" 6L, 203C 6,0-1 All
```

Unidade 2

1. Utilizamos em nosso estudo de caso apenas um dos métodos disponíveis no Javascript para utilizar uma classe que foi definida em outro arquivo. Existe um outro método utilizando a palavra-chave “import” do Javascript. Pesquise na Internet sobre ela, experimente-a em um programa de exemplo e informe aqui qual dos formatos é melhor: “require” ou “import”.

Solução:

O mais importante neste item está implícito: é o interesse no estudante pela pesquisa de forma autodidata, fundamental para a sobrevivência de qualquer desenvolvedor hoje dia. A resposta mais correta e atual é que a palavra-chave import é mais recente no Javascript e permite uso mais granular, por exemplo, por ele é possível importar somente uma de várias classes que estejam declaradas em um arquivo. Então por ser mais recente e permitir mais granularidade, é melhor que a opção “require”. Mas não se deve penalizar o estudante se ele construir uma argumentação que o faça acreditar que o “require” é melhor também - há quem o prefira exatamente pelo motivo contrário ao do “import”: por ser mais velho, é mais

suportado em diferentes versões do Javascript.

2. Vamos adicionar algumas bibliotecas em nosso arquivo package.json, elas serão úteis na próxima unidade. Pedimos que no diretório do projeto onde gerou seu próprio package.json, faça a instalação das seguintes bibliotecas:

- a. body-parser
- b. ejs
- c. csv

Solução:

Para cada item acima, o/a estudante precisa apenas executar os comandos de instalação para trazer cada biblioteca referida para o projeto (requer conexão com a Internet):

- npm install body-parser
- npm install ejs
- npm install csv

Unidade 3

1. Altere o programa para que o arquivo de dados de pessoas seja configurável numa variável de ambiente.

Solução:

```
require('dotenv').config();
```

```
const fs = require("fs");
```

```
const papa = require("papaparse");
```

```
const options = {header: true, dynamicTyping: true};
```

```
const parseStream = papa.parse(papa.NODE_STREAM_INPUT, options);
```

```
let data = [];
```

```
var admin_existe = false;

const APP_PORT = process.env.APP_PORT;

// Resolução (1): criar uma variável que carregue a variável de ambiente.

const ARQ_PESSOAS = process.env.ARQ_PESSOAS;

// Resolução (2): Fazer com que a leitura utilize a variável.

const dataStream = fs.createReadStream(ARQ_PESSOAS);

console.log(`Executando app na porta ${APP_PORT}`);

// Realiza a leitura do arquivo e passa o resultado para 'parseStream'.

dataStream.pipe(parseStream);

// Cada pedaço de dado lido alimenta a variável "data"

parseStream.on("data", chunk => {

  data.push(chunk);

});

// Ao finalizar a leitura, verificamos se existe um usuário "admin"

parseStream.on("finish", () => {

  for (var i = 0; i < data.length; i++) {

    user = data[i];

    if (user.Login == 'admin') {

      admin_existe = true;

      break;

    }

  }

  if (admin_existe == true) {

    // Admin existe, nada precisa ser feito.

    console.log('Usuário admin encontrado!');

  } else {
```

```

// Admin inexistente, precisamos criá-lo.

user_admin = {

  Login: 'admin',

  Nome: 'Administrador',

  Sobrenome: null,

  Senha: 'fullture@4545'

};

data.push(user_admin);

csv_data = papa.unparse(data);

// Resolução (3): utilizar a variável para gravar o arquivo.

fs.writeFileSync(ARQ_PESSOAS, csv_data);

console.log('Criação do usuário admin realizado!');

}

});

# Resolução (4): alterar o arquivo .env (dentro do diretório src/)

# Arquivo .env (dentro do diretório src/)

APP_PORT=4000

ARQ_PESSOAS="./pessoas.csv"

```

2. Modifique novamente o programa para que mostre em que modo (desenvolvimento ou produção) ele está sendo executado.

Solução:

```

require('dotenv').config();

const fs      = require("fs");

const papa    = require("papaparse");

const options = {header: true, dynamicTyping: true};

```

```

const parseStream = papa.parse(papa.NODE_STREAM_INPUT, options);

let data      = [];

var admin_existe = false;

const APP_PORT  = process.env.APP_PORT;

// Resolução (1): criar uma variável que carregue a variável de ambiente.

const NODE_ENV = process.env.NODE_ENV;

// Resolução (2): Fazer com que a leitura utilize a variável.

console.log(`Programa executando em modo ${NODE_ENV}`);

// O resto do programa não importa para esse exercício.

// ...

# Resolução (3): alterar o arquivo .env (dentro do diretório src/)

# Arquivo .env (dentro do diretório src/)

APP_PORT=4000

ARQ_PESSOAS="./pessoas.csv"

  • NODE_ENV="production"

```

Unidade 4

1. Tente incrementar o experimento desta unidade gerando um arquivo de dados para os mentores do sistema. Esse arquivo de dados deve conter um ID, o nome completo do mentor, seu endereço de e-mail junto ao assunto de especialidade. Gere um link na página raiz para visualizar a lista de mentores. Ao clicar nesse link, uma página deve mostrar a lista contendo apenas o nome de todos os mentores. Como no caso dos alunos, cada item da lista gerada deve também ser um link que leve o aluno para uma página de detalhes, mostrando todos os dados cadastrados para ele.

Solução:

O(A) estudante deve:

- Criar um arquivo de dados JSON para os mentores em /dados, contendo pelo menos um mentor.
- Alterar no arquivo “index.js” o método que trata a página raiz para inserir um link que exiba a lista de mentores.
- Referenciar o arquivo de dados criado para os mentores dentro do programa principal.
- Gerar duas novas funções Javascript, uma para mostrar a lista de mentores e outra para mostrar os detalhes de cada mentor.

2. A página inicial do nosso sistema oferece um link para criar um novo aluno que não está funcionando. Responda:

- a. Quais *requests* e verbos precisam ser construídos para que seja possível **cadastrar um novo aluno** em nosso arquivo JSON?
- b. Como seria a assinatura da função para tratar a requisição que pode criar um novo registro de aluno em nosso arquivo “index.js”? Lembre-se que estamos utilizando a framework “Express”.

Solução:

É preciso criar uma requisição GET para obter uma página com um formulário de cadastro e mais uma requisição com o verbo POST para enviar a requisição que vai criar o aluno. Vale destacar que o(a) estudante que souber que o formulário de registro precisa de uma variável “method” com o valor “post”.

O importante é que o(a) estudante use a função “post”, conforme o seguinte exemplo: “app.post('/', function (req, res) { }”.