Desafio Serasa Ricardo Lindner

Importante: minha área de atuação atual não é a programação, portanto percebi neste programa do Serasa em conjunto com a Proway uma ótima oportunidade para iniciar na Tecnologia da Informação. O Javascript é a linguagem que eu comecei a estudar recentemente, e decidi utilizá-la por isso.
O Desafio proposto foi além das minhas capacidades e conhecimento técnico, por mais que eu tenha buscado informação em vídeos e bancos de conhecimento disponíveis na internet, dentro do tempo que tinha disponível dadas as minhas demais responsabilidades. Avancei até certo ponto no desafio, **contudo sem concluí-lo**, e submeto a seguir a documentação contendo os registros das etapas que finalizei.

Requisitos mínimos:

- Banco de dados MySQL;
- Ferramenta de edição SQL (utilizado MySQL Workbench);
- node.js (https://nodejs.org/en/download/);
- Editor de códigos (utilizado SublimeText3);
- Navegador (utilizado Chrome).

Banco de dados MySQL:

- Criado Banco de Dados chamado desafioserasa através do comando CREATE DATABASE no cmd;
- Comando SQL para criar tabela Empresa:

```
CREATE TABLE IF NOT EXISTS Empresa(IDEmpresa int AUTO_INCREMENT NOT NULL PRIMARY KEY, nome varchar(100) NOT NULL,CNPJ INT NULL DEFAULT NULL, ClassificacaoEmpresa INT NOT NULL DEFAULT 50);
```

- Comando SQL para criar tabela de Operações Financeiras:

```
CREATE TABLE OperacoesFinanceiras (IDOperacao BIGINT AUTO_INCREMENT NOT NULL PRIMARY KEY, IDEmpresa INT NOT NULL, MesOperacao DATE NOT NULL, QuantidadeNotasFiscais INT NOT NULL, QuantidadeDebitos INT NOT NULL);
```

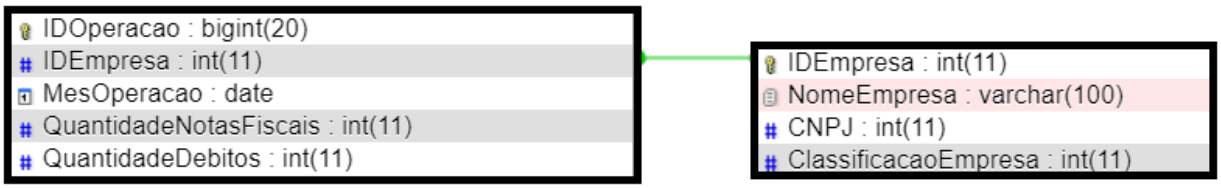
- Comando SQL para vincular chave estrangeira:

```
ALTER TABLE OperacoesFinanceiras ADD CONSTRAINT
fk_IDEmpresa_Operacao
FOREIGN KEY (IDEmpresa) REFERENCES Empresa (IDEmpresa)
ON DELETE RESTRICT ON UPDATE RESTRICT;
```

- Comando SQL para Popular Tabela Empresa:

```
INSERT INTO Empresa
VALUES (DEFAULT, 'Empresa01',0, DEFAULT),
       (DEFAULT, 'Empresa02',0, DEFAULT),
       (DEFAULT, 'Empresa03',0, DEFAULT),
       (DEFAULT, 'Empresa04',0, DEFAULT),
       (DEFAULT, 'Empresa05',0, DEFAULT),
       (DEFAULT, 'Empresa06',0, DEFAULT),
       (DEFAULT, 'Empresa07',0, DEFAULT),
       (DEFAULT, 'Empresa08',0, DEFAULT),
       (DEFAULT, 'Empresa09',0, DEFAULT),
       (DEFAULT, 'Empresa10',0, DEFAULT),
       (DEFAULT, 'Empresa11',0, DEFAULT),
       (DEFAULT, 'Empresa12',0, DEFAULT),
       (DEFAULT, 'Empresa13',0, DEFAULT),
       (DEFAULT, 'Empresa14',0, DEFAULT),
       (DEFAULT, 'Empresa15',0, DEFAULT),
       (DEFAULT, 'Empresa16',0, DEFAULT),
       (DEFAULT, 'Empresa17',0, DEFAULT),
       (DEFAULT, 'Empresa18',0, DEFAULT),
       (DEFAULT, 'Empresa19',0, DEFAULT),
       (DEFAULT, 'Empresa20',0, DEFAULT),
       (DEFAULT, 'Empresa21',0, DEFAULT)
```

- Estrutura de dados:



Node.js:
O objetivo de utilizar o Node.js era permitir a conexão com o banco através de recursos back end no javascript, linguagem que tenho maior contato, mesmo sendo uma ferramenta majoritariamente front end.
Com o Node.js foram utilizados as ferramentas: Express, nodemon, sequelize, mysql2, express-handlebars e body-parser.

- Comando para instalar o Express:

C:\Users\rilin\OneDrive\Área de Trabalho\Projeto Serasa>npm install express --save

- Comando para instalar o nodemon:

C:\Users\rilin\OneDrive\Área de Trabalho\Projeto Serasa>npm install nodemon -g

- Comando para instalar o sequelize:

C:\Users\rilin\OneDrive\Área de Trabalho\Projeto Serasa>npm install --save sequelize

- Comando para instalar o mysql2:

C:\Users\rilin\OneDrive\Área de Trabalho\Projeto Serasa>npm install --save mysql2

- Comando para instalar o express-handlebars:

C:\Users\rilin\OneDrive\Área de Trabalho\Projeto Serasa>npm install --save express-handlebars

- Comando para instalar o body-parser:

C:\Users\rilin\OneDrive\Área de Trabalho\Projeto Serasa>npm install --save body-parser

Códigos:
Os códigos foram organizados em pastas diferenciadas para os tipo de arquivo “.css”, “.html” e “.js”, com exceção do arquivo “index.html” que ficou disponibilizado na pasta raiz.

- Arquivos “.css”
  Nesta pasta são disponibilizados 2 arquivos: “reset.css” e “index.css”
  - index.css: Esse arquivo concentra toda a formatação da página;
  - reset.css: Arquivo que reseta completamente o css da página, para evitar problemas com pré-configurações do navegador.
- Arquivos “.html”
  Os arquivos html utilizados são “dados-fianceiros.html”, “index.html” e “teste-unitario.html”. Cada um dos arquivos é uma página da solução.
  Importante: devido aos critérios de cálculo, que determinam que as notas e débitos devem ser calculadas individualmente uma após a outra, o critério de

arredondamento pra cima para débitos não permite que o ranking da empresa seja inferior a 24%.

- dados-fianceiros.html: esta é a página que disponibilizaria o import dos dados financeiros para a base de dados;
- index.html: **este arquivo é a página inicial** e principal, onde seria disponibilizado o ranking dinâmico das empresas;
- teste-unitario.html: Por fim a guia de teste unitário é aonde é possível validar as regras de cálculo. **Importante:** devido aos critérios de cálculo, que determinam que as notas e débitos devem ser calculadas individualmente uma após a outra, o critério de arredondamento pra cima para débitos não permite que o ranking da empresa seja inferior a 24%.

- Arquivos “.js”
  Os arquivos html utilizados são “classificacao.js”, “db.js”, “empresa.js” “formulario.js” “index.js”.
  - classificacao.js: arquivo responsável por apurar a classificação das empresas.

```
/*Define constantes utilizadas para apurar a classificação da empresa
conforme as notas e débitos*/
const INDICE_AUMENTO_NOTA = 1.02;
const INDICE_REDUCAO_NOTA = 0.96;

/*Instancia a Variável Classificação de Empresas, utilizada nas funções
de cálculo*/
var classificacaoEmpresa = 0;


/*Função que calcula a classificação da empresa por nota fiscal*/
function processaNotasFiscais() {
	if (classificacaoEmpresa * INDICE_AUMENTO_NOTA > 100) {
		return classificacaoEmpresa = 100;
	} else {
		return classificacaoEmpresa =
Math.trunc(classificacaoEmpresa * INDICE_AUMENTO_NOTA);
	}
}

/*Função que calcula a classificação da empresa por débitos*/
/*Dado critério de arredondamento para cima o menor índice possível será
24%*/
function processaDebitos() {
	if (classificacaoEmpresa * INDICE_REDUCAO_NOTA < 1) {
		return classificacaoEmpresa = 1;
	} else {
```

```
                return classificacaoEmpresa =
Math.ceil(classificacaoEmpresa * INDICE_REDUCAO_NOTA);

        }
}

/*Função que calcula a classificação geral da empresa considerando o
número de notas e número de débitos do período*/
function calculaClassificacao(ranking, notas, debitos) {
        classificacaoEmpresa = ranking;
        for (var i = 0; i < notas; i++) {
                processaNotasFiscais();
        }

        for (var i = 0; i < debitos; i++) {
                processaDebitos();
        }
        return classificacaoEmpresa;
}
```

- db.js: arquivo responsável pela conexão com o banco de dados local.

```
/*Módulo para conectar no Banco de Dados local*/
const Sequelize = require("sequelize")

// Conexão com o banco de dados MySQL
const sequelize = new Sequelize("desafioserasa", "root",
"BDsenha25522020*", {
        host: "localhost",
        dialect: "mysql"
})

module.exports = {
        Sequelize: Sequelize,
        sequelize: sequelize
}
```

- empresa.js: arquivo responsável pela extração e conversão da tabela empresas do banco de dados.

```
/*Módulo que utiliza o módulo de conexão com o BD para importar a tabela
empresa*/

const db = require("./db")

const empresa = db.sequelize.define("empresa", {
        IDEmpresa: {
                type: db.Sequelize.INTEGER
        },
        Nome: {
                type: db.Sequelize.STRING
        },
```

```
        CNPJ: {
                type: db.Sequelize.INTEGER
        },
        ClassificacaoEmpresa: {
                type: db.Sequelize.INTEGER
        }
})

module.exports = empresa
```

- formulario.js: arquivo que contém o código para calcular os testes unitários.

```
/*Variável que busca informações no formulário em tela*/
var botaoCalcular = document.querySelector("#calcula-classificacao");

/*Função que concatena demais funções para gravar dados de formulário na
tabela*/
botaoCalcular.addEventListener("click", function (event) {
        event.preventDefault();

        var formulario = document.querySelector("#form-calcula");
        var empresa = obtemEmpresaDoFormulario(formulario);

        console.log(empresa);

        var erros = validaEmpresa(empresa);
        console.log(erros);
        if (erros.length > 0) {
                exibeMensagensDeErro(erros);
                return;
        }

        adicionaEmpresaNaTabela(empresa);

        formulario.reset();

        var mensagensErro = document.querySelector("#mensagens-erro");
        mensagensErro.innerHTML = "";
});

/*Função que grava dados de formulário na tabela*/
function adicionaEmpresaNaTabela(empresa) {
        var empresaTr = montaTr(empresa);
        var tabela = document.querySelector("#tabela-empresas");
        tabela.appendChild(empresaTr);
}

/*Função que captura os dados do formulário*/
function obtemEmpresaDoFormulario(formulario) {
        var empresa = {
                nome: formulario.empresa.value,
```

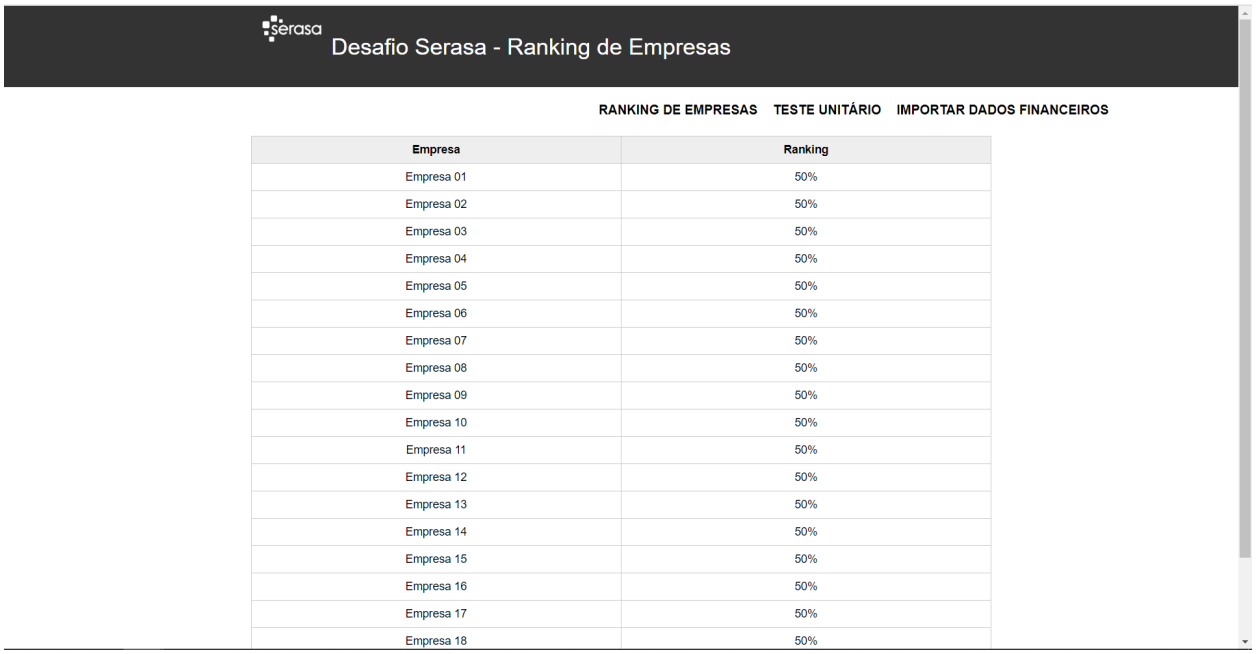
```
                ranking: formulario.ranking.value,
                notas: formulario.notas.value,
                debitos: formulario.debitos.value,
                rankingFinal:
(calculaClassificacao(formulario.ranking.value, formulario.notas.value,
formulario.debitos.value) + "%")
        }
        return empresa;
}

/*Função que monta Tr do HTML automaticamente*/
function montaTr(empresa) {
        var empresaTr = document.createElement("tr");
        empresaTr.classList.add("empresa");

        empresaTr.appendChild(montaTD(empresa.nome, "info-nome"));
        empresaTr.appendChild(montaTD(empresa.ranking, "info-ranking"));
        empresaTr.appendChild(montaTD(empresa.notas, "info-notas"));
        empresaTr.appendChild(montaTD(empresa.debitos, "info-debitos"));
        empresaTr.appendChild(montaTD(empresa.rankingFinal,
"info-rankingFinal"));

        return empresaTr;
}

/*Função que monta Td do HTML automaticamente*/
function montaTD(dado, classe) {
        var td = document.createElement("td");
        td.textContent = dado;
        td.classList.add(classe);

        return td;
}

/*Função que valida as informações do ranking preenchidas no teste
unitário*/
function validaRankingInicial(ranking) {
        if (ranking >= 1 && ranking <= 100) {
                return true;
        } else {
                return false;
        }
}

/*Função que valida as informações de notas preenchidas no teste
unitário*/
function validaNotas(notas) {
        if (notas >= 0) {
                return true;
        } else {
```

```
			return false;
		}
}

/*Função que valida as informações de débitos preenchidas no teste
unitário*/
function validaDebitos(debitos) {
		if (debitos >= 0) {
			return true;
		} else {
			return false;
		}
}

/*Função que valida em conjunto as todas as informações necessárias o
teste unitário*/
function validaEmpresa(empresa) {

		/*Define a variável erros como um array vazio*/
		var erros = [];

		/*Se a validação do ranking da empresa resultar em falso registra
mensagem de erro no array*/
		if (!validaRankingInicial(empresa.ranking))
			erros.push("O Ranking Inicial informado não é válido!");
		/*Se a validação das notas da empresa resultar em falso registra
mensagem de erro no array*/
		if (!validaNotas(empresa.notas))
			erros.push("O número de Notas informado não é válido!")

		/*Se a validação dos débitos da empresa resultar em falso registra
mensagem de erro no array*/
		if (!validaDebitos(empresa.debitos))
			erros.push("O número de Débitos informado não não é
válido!")

		/*Verifica se o campo nome da empresa está em branco no
formulário*/
		if (empresa.nome.length == 0) {
			erros.push("O nome da empresa não pode ficar em branco!");
		}

		/*Verifica se o campo ranking está em branco no formulário*/
		if (empresa.ranking.length == 0) {
			erros.push("O ranking inicial da empresa não pode ficar em
branco!");
		}

		/*Verifica se o campo notas está em branco no formulário*/
		if (empresa.notas.length == 0) {
```

```
                erros.push("O número de notas no mês não pode ficar em
branco!");
        }

        /*Verifica se o campo débitos está em branco no formulário*/
        if (empresa.debitos.length == 0) {
                erros.push("O número de débitos no mês não pode ficar em
branco!");
        }

        return erros;
}

function exibeMensagensDeErro(erros) {
        var ul = document.querySelector("#mensagens-erro");
        ul.innerHTML = "";

        erros.forEach(function (erro) {
                var li = document.createElement("li");
                li.textContent = erro;
                ul.appendChild(li);
        });
}
```

A Aplicação:

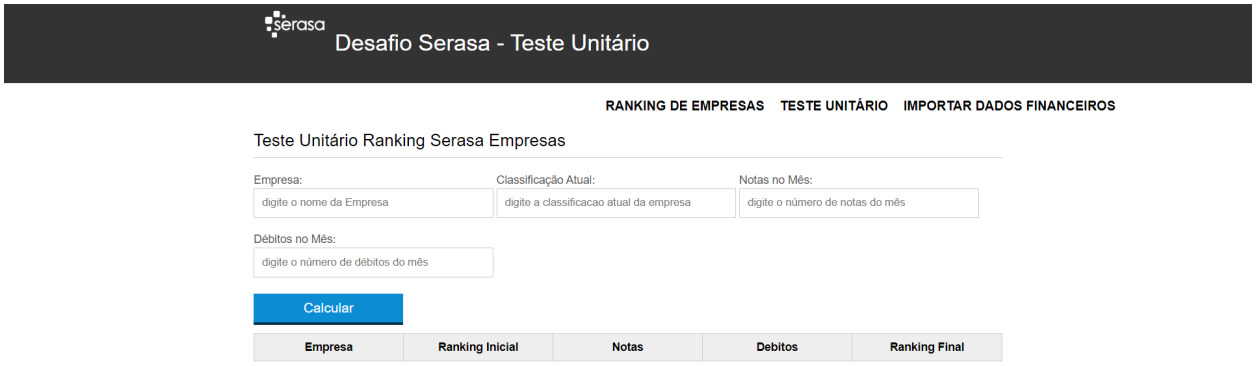
A aplicação é iniciada com o arquivo “index.html” disponibilizado na pasta raíz. Ao abrir o arquivo será carregada a página contendo o ranking das empresas:



Importante: conforme mencionado no início do documento, infelizmente este ranking está fixado no código e não dinâmico conforme solicitado nos requisitos.

A navegação na aplicação se dá pelos links existentes:

Em Teste Unitário é possível lançar empresas aleatórias e informar o ranking inicial, número de notas e número de débitos para calcular o ranking, testando se a aplicação está aplicando corretamente os critérios estabelecidos:



A última tela disponível é Importar dados, que permite que você selecione um arquivo no seu computador: