Manual técnico

SIMULADOR DE CALL CENTER

Mishell Monroy
PRÁCTICA 1, LENGUAJES FORMALES DE PROGRAMACIÓN
202404409

Índice

Introducción

Este manual, explica la creación del código, para la elaboración de una aplicación para gestionar el movimiento de llamadas de un Call Center, donde se pueda generar ciertas funcionalidades como leer un archivo csv, generar reportes en un documento HTML, usando funcionalidades de Node, sin utilizar librerías de Java Script.

Objetivos:

Brindar una idea clara de cómo fue creado el programa, para que cualquier otra persona pueda interpretar de una buena manera el código creado, incluso es viable ver este manual para cuando se le desea hacer una actualización el programa, se sabe dónde se debe de hacer, debido a que en este documento se explica de forma clara y ordenada las funciones del código.

Dirigido

Está dirigido a todas aquellas personas que deseen modificar o actualizar el programa, también para quienes deseen un programa como este, para que sepan cómo deben leer el código y como es su funcionamiento, incluso está dirigido a el programador en sí, ya que de esta forma puede saber de una manera más clara y ordenada que es y donde está el código que desea implementar o actualizar.

Especificación técnica

Requisitos de hardware

- Disco duro capaz de correr el programa, a partir de 250 GB
- Procesador Core i5 o superior
- 4GB de memoria RAM como mínimo.

Requisitos de Software

- Debe tener instalado algún editor de código fuente, para usar el programa directamente, Computadora de escritorio o portátil, se recomienda utilizar Visual Studio Code.
- Node, version v22.17.1
- Instalar previamente el Java

Lógica del programa

Diccionario de Import utilizadas:

I. import fs from 'fs'; (sirve para importar el módulo fs (File System) en Node.js, que permite leer, escribir, modificar y gestionar archivos y carpetas en el sistema)

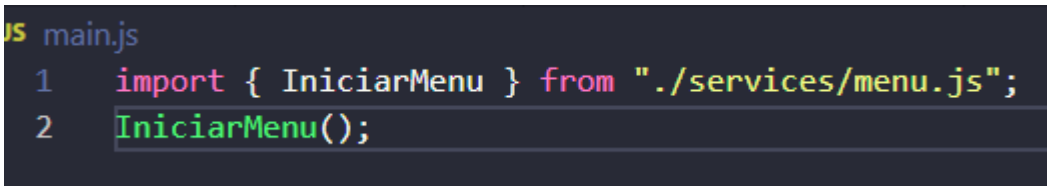
II. import path from "path"; (sirve para trabajar con rutas de archivos y directorios de forma segura y multiplataforma en Node.js.)

III. import readline from "readline"; (Es un módulo nativo de Node.js que te permite leer entradas del usuario línea por línea, como si estuvieras haciendo un mini chatbot o un formulario en consola.)

Método main

Main

En el main únicamente se tiene la inicialización de la aplicación, solamente se usa para iniciar el programa, manda a llamar a la clase “Menu” que muestra el menú inicial, donde se puede decidir qué operación desea realizar.



Lógica del programa

Para este programa se utilizó Programación Orientada a objetos, así que se utilizaron varias clases, para poder darle un funcionamiento bien estructurado al programa.

Clases utilizadas

1. CallCenter.js
2. CargarArchivo.js
3. ExportarHistorial.js
4. Menú.js
5. Operaciones.js
6. Parser.js

CallCenter

En esta clase se define al objeto CallCenter, donde se especifican los atributos del mismo y algunos métodos que al inicializar el constructor se espera que realice.

```
export default class CallCenter{
    constructor(id_operador,nombre_operador,estrellas,id_cliente,nombre_cliente){
        this.id_operador = id_operador;
        this.nombre_operador = nombre_operador;
        this.estrellas = estrellas;
        this.id_cliente = id_cliente;
        this.nombre_cliente = nombre_cliente;

        this.estrellas = this.contarEstrellas();
        this.rendimiento = this.clasificarOperador();

    }
    contarEstrellas() {
    return this.estrellas.split(";").filter(e => e.toLowerCase() === "x").length;
    }

    clasificarOperador() {
        if (this.estrellas <= 1) return "Malo";
        if (this.estrellas <= 3) return "Regular";
        if (this.estrellas <= 5) return "Buenisimo";
        return "ninguno";
    }

}
```

Se definen los atributos del centro de llamadas y luego aprovechando el recurso de las Estrellas, se escriben 2 funciones propias del objeto, para contar el número de estrellas y poder devolver un número entero al leer el archivo .csv y una función para clasificar al operador en sus 3 rangos según el número de estrellas registradas.

Cargar Archivo

Lo que hace esta clase es utilizar import fs from "fs" y import path from "path", para poder leer el archivo .csv y poder modificarlo, lo que hace es leer el archivo y guardarlo en una lista, para poder gestionarlo, lo guarda como objeto CallCenter, mandando el atributo a donde corresponde y asi hacer mas flexible su utilización.

```
import fs from "fs";
import path from "path";
import CallCenter from "../Practica/CallCenter.js";
/**
 * Función para leer un archivo CSV y mostrarlo en consola
 * @param {string} rutaArchivo - Ruta del archivo CSV
 */

export function LeerArchivo(rutaArchivo) {
  try {
    const contenido = fs.readFileSync(path.resolve(rutaArchivo), "utf8"); //lee el archivo
    const lineas = contenido.split("\n").map(l => l.trim()).filter(l => l.length > 0); //
    const encabezados = lineas[0].split(",").map(e=> e.trim()); //extrae los encabezados

    // Mapea cada línea de datos a un objeto
    const datos = lineas.slice(1).map(linea => {
      const valores = linea.split(",").map(e => e.trim());
      let fila = {};
      encabezados.forEach((col, i) => {
        fila[col] = valores[i];
      });
```

Los datos tienen la siguiente estructura para ser imprimidos en consola, en forma de una tabla.

```
console.log("\nRegistros de llamadas:", rutaArchivo);
console.table(datos.map(r => ({
  id_operador: r.id_operador,
  nombre_operador: r.nombre_operador,
  estrellas: `${r.estrellas}`,
  id_cliente: r.id_cliente,
  nombre_cliente: r.nombre_cliente
})));
```

Exportar Historial

Utiliza la información y funciones de otras clases, para crear un archivo html y exportar los datos en ciertas clasificaciones.

```
import fs from 'fs';
import { obtenerOperadores, obtenerClientes, porcentaje } from './Operaciones.js';

export function ReporteHTML(archivo){
    const Operadores = obtenerOperadores(archivo); // ← Map con ID y nombre
    const Clientes = obtenerClientes(archivo); // ← Map con ID y nombre

    let html = `
        <!DOCTYPE html>
        <html lang="es">
        <head>
            <meta charset="UTF-8">
            <title>Historial de Llamadas</title>
            <style>
                table { border-collapse: collapse; width: 100%; }
                th, td { border: 1px solid #ccc; padding: 8px; text-align: left; }
                th { background-color: #f2f2f2; }
            </style>
        </head>
        <body>
```

Se declaran las constantes Operadores y Clientes, para poder acceder a esas funciones contenidas por “Operaciones” y poder imprimir solo los operadores y clientes sin repetirlos, para las cuales ambas estructuras son la siguiente.

```
            <h1>Listado de Operadores</h1>
            <table>
                <tr>
                    <th>ID Operador</th>
                    <th>Nombre Operador</th>
                </tr>
            `;

Operadores.forEach((nombre, id) => {
    html += `
        <tr>
            <td>${id}</td>
            <td>${nombre}</td>
        </tr>
        `;
    });

    html += `
        </table>
```

Menú

Es con quien interactúa el usuario, se presentan las opciones a elegir y tiene el manejo de acceder a cada método y que se ejecuten, se utiliza ReadLine que es un tipo Scanner en Java, para poder leer la opción elegida por el usuario.

```
import readline from "readline";
import { LeerArchivo } from "./CargarArchivo.js";
import { ReporteHTML } from "./ExportarHistorial.js";
import { contarClasificaciones, porcentajeClasificaciones } from "./Operaciones.js";

export function IniciarMenu() {
  const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
  });

  let archivo = []; // almacena los registros cargados
```

Interfaz del menú:

```
function Menu() {
  console.log("\n----------------- CONTROL DE CALL CENTER -----------------");
  console.log("| 1. Cargar Registros de Llamadas                          |");
  console.log("| 2. Generar reportes (Exportar Historial de Llamadas,     |");
  console.log("|    Listado de Operadores, Listado de Clientes y          |");
  console.log("|    Rendimiento de Operadores)                            |");
  console.log("| 3. Mostrar Porcentaje de Clasificación de Llamadas       | ");
  console.log("| 4. Mostrar Cantidad de Llamadas por Calificación         | ");
  console.log("| 5. Salir                                                 |");
  console.log(" ----------------------------------------------------------");
  rl.question("-> Seleccione una opción: ", handleMenuOption);
}
```

Lógica de las operaciones realizadas, usando un switch.

```
function handleMenuOption(opcion) {
  switch (opcion.trim()) {
      case "1":
          rl.question("Ingrese el nombre del archivo CSV: ", (nombreArchivo) => {
          archivo = LeerArchivo(nombreArchivo); // ← Aquí guardas los datos correctamente
          Menu();
          });
          break;
      case "2":
          if (archivo.length === 0) {
            console.log("No hay datos cargados. Por favor, cargue un archivo primero.");
          } else {
            ReporteHTML(archivo);
          }
            Menu();
          break;
```

Operaciones

En esta clase se desarrollan las funciones que varias clases van a necesitar para su correcto funcionamiento, esto es para hacer mas ordenado el código y tener todas las funciones en una misma carpeta, listas solo para ser llamadas, contiene funciones como:

Obtener Operadores

Busca solo los operadores dentro del archivo csv, para imprimirlos en el html y que no se repitan.

```
export function obtenerOperadores(instancias) {
    const mapa = new Map();
    instancias.forEach(obj => {
        if (!mapa.has(obj.id_operador)) {
            mapa.set(obj.id_operador, obj.nombre_operador);
        }
    });
    return mapa;
}
```

Obtener Clientes

Busca solo los operadores dentro del archivo csv, para imprimirlos en el html y que no se repitan.

```
export function obtenerClientes(instancias) {
    const mapa = new Map();
    instancias.forEach(obj => {
    if (!mapa.has(obj.id_cliente)) {
        mapa.set(obj.id_cliente, obj.nombre_cliente);
    }
    });
    return mapa;
}
```

Contar operadores

Cuenta la cantidad de operadores solos que hay, y regresa el total de filas que tiene el archivo y el numero de operadores, para ser usados posteriormente en otros cálculos.

```
export function contarOperadores(registros) {
    const totalFilas = registros.length;
    const frecuenciaPorOperador = {};

    registros.forEach(registro => {
    const nombre = registro.nombre_operador;
    if (!frecuenciaPorOperador[nombre]) {
        frecuenciaPorOperador[nombre] = 1;
    } else {
        frecuenciaPorOperador[nombre]++;
    }
    });

    return { totalFilas, frecuenciaPorOperador };
}
```

Porcentaje

Usa la función previamente definida para hacer un cálculo aritmético y determinar el porcentaje del rendimiento por cada operador

```
export function porcentaje(obtenerOperadores, totalFilas) {
    const porcentajePorOperador = {};
    for (const [nombre, frecuencia] of obtenerOperadores) {
        porcentajePorOperador[nombre] = ((frecuencia / totalFilas) * 100).toFixed(2);
    }
    return porcentajePorOperador;
}
```

Contar Clasificaciones

Cuenta que tantas llamadas están calificadas como Bueno, Regular, Malo o en su defecto ninguno.

```
export function contarClasificaciones(registros) {
    const conteo = {
    Buenisimo: 0,
    Regular: 0,
    Malo: 0,
    ninguno: 0
    };

    registros.forEach(r => {
    const tipo = r.rendimiento;
    if (conteo.hasOwnProperty(tipo)) {
        conteo[tipo]++;
    } else {
      conteo.nadota++; // por si hay un valor inesperado
    }
    });
```

Porcentaje de Clasificaciones

Según el conteo previamente calculado, se hace una operación aritmética para calcular el porcentaje de calificaciones según su rango, es decir un porcentaje de que tantas llamadas son calificadas en el rango establecido.

```
export function porcentajeClasificaciones(registros) {
  const total = registros.length;
  if (total === 0) {
    console.log("No hay registros para calcular porcentajes.");
    return;
  }

  const conteo = contarClasificaciones(registros);
  console.log("-------- Porcentaje de clasificación de operadores --------");
  Object.entries(conteo).forEach(([tipo, cantidad]) => {
    const porcentaje = ((cantidad / total) * 100).toFixed(2);
    console.log(`- ${tipo}: ${porcentaje}%`);
  });
}
```

Parser.js

Sirve para parcear el documento, es decir, asegurarse que en los campos donde van solo enteros, por ejemplo, solo se ingresen números enteros o convertirlo a entero o cadena str.