MANUAL TECNICO

USAC CALL-CENTER

Lenguajes Formales y de Programación

José Fernando Ramirez Ambrocio

Carnet: 202400195

ÍNDICE

MANUAL TÉCNICO

I. Introducción:

Este manual técnico proporciona una descripción detallada de la arquitectura, componentes, flujo de trabajo y configuración del Simulador de CallCenter, una aplicación desarrollada en JavaScript (Node.js) que permite procesar registros de llamadas, generar reportes y analizar el rendimiento de operadores.
El sistema está diseñado para funcionar en consola, sin el uso de librerías externas, cumpliendo con los principios de Programación Orientada a Objetos (POO) y manipulación de archivos con funciones nativas del lenguaje.
Este documento está dirigido a desarrolladores, auxiliares de cátedra y personal técnico que deseen entender, mantener o extender la aplicación.

II. Objetivos

El objetivo principal de este manual es proporcionar una guía detallada sobre el funcionamiento del programa, permitiendo que otros desarrolladores puedan analizar y modificar el código de manera eficiente.

Además, se busca:

- Facilitar la comprensión de la estructura del sistema.
- Documentar el flujo de datos desde la carga del archivo hasta la generación de reportes.
- Mostrar el uso de estructuras de datos y POO en JavaScript.
- Cumplir con los requisitos de la práctica de LFP.

III. Dirigido

Este documento está orientado a:

Estudiantes de programación interesados en la estructura de un simulador de CallCenter.

Auxiliares de cátedra encargados de la revisión presencial.

Cualquier persona que desee comprender cómo se ha implementado la lógica de esta aplicación en JavaScript.

IV. Especificación técnica

4.1. Requisitos de Hardware

- Computadora de escritorio o portátil
- Memoria RAM: 2GB o superior
- Espacio en disco: 100MB o superior
- Procesador: Intel Core i3 o equivalente
- Resolución de pantalla: 1024 x 768 px o superior

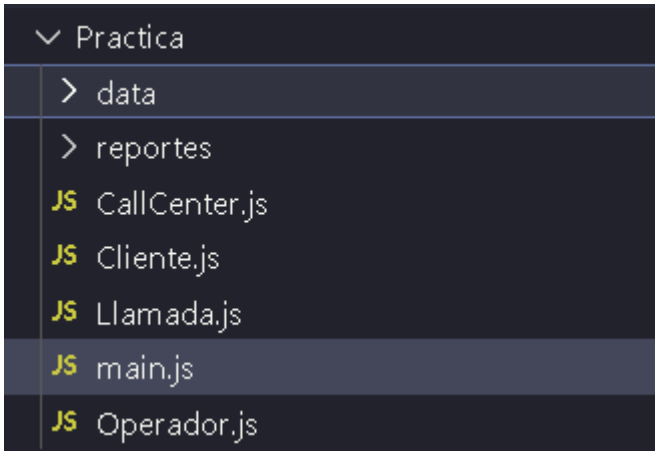
4.2. Requisitos de Software

- Sistema Operativo: Windows 7 o superior, macOS o Linux
- Node.js Version 14 o superior.
- Visual Studio Code, Sublime Text, etc.
- Librerías utilizadas solo módulos nativos: fs, readline.

V. Lógica del Programa

5.1 Estructura del Programa:

El programa se organiza en una estructura modular basada en POO, con las siguientes carpetas y archivos:



5.2 Flujo del Programa:

1. El usuario ejecuta node main.js.
2. Se muestra el menú principal.
3. Al seleccionar "Cargar Registros", el usuario ingresa la ruta del archivo .csv.
4. El programa lee el archivo con fs.readFileSync, lo divide por líneas y campos.
5. Para cada línea:
    - Crea una instancia de Llamada

- Extrae id operador, nombre operador, estrellas (contando las x)
- Crea instancias de Operador y Cliente
- Guarda todo en CallCenter

6. Al generar reportes:
   - Se crean tablas HTML con estilos básicos
   - Se escriben con fs.writeFileSync
7. Al mostrar estadísticas:
   - Se calculan porcentajes y conteos
   - Se muestran en consola

5.3 Descripción de Funcionalidades:

Funcionalidad	Descripción
Cargar Llamadas	Lee archivo .csv, procesa estrellas (x;x;0;0;0) y guarda en memoria
Historial de Llamadas	Historial de Llamadas
Genera HTML con todas las llamadas, operadores, clientes y clasificación	Genera HTML con todas las llamadas, operadores, clientes y clasificación
Listado de Operadores	Listado de Operadores
Genera HTML con ID y nombre de cada operador	Genera HTML con ID y nombre de cada operador
Listado de Clientes	Listado de Clientes
Genera HTML con ID y nombre de cada cliente	Genera HTML con ID y nombre de cada cliente

VI. Clases y Métodos

6.1 Clases Principales:

6.1.1. CallCenter

Clase principal que gestiona todos los datos y reportes.

```
Practica > JS CallCenter.js > ...
1   const fs = require('fs');
2   const Llamada = require('./Llamada');
3
4   class CallCenter {
5       constructor() {
6           this.llamadas = [];
7           this.operadores = new Map();
8           this.clientes = new Map();
9       }
10
11      cargarLlamadas(ruta) {
12          try {
13              const data = fs.readFileSync(ruta, 'utf8');
14              const lineas = data.trim().split('\n');
15
16              if (lineas.length < 2) {
17                  console.log("El archivo está vacío o no tiene datos.");
18                  return;
19              }
20
21              this.llamadas = [];
22              this.operadores.clear();
23              this.clientes.clear();
24
25              for (let i = 1; i < lineas.length; i++) {
26                  const linea = lineas[i].trim();
27                  if (!linea) continue;
28
29                  const partes = linea.split(',');
30                  if (partes.length !== 5) {
31                      console.warn("Formato incorrecto (no 5 columnas):", linea);
32                      continue;
33                  }
```

6.1.2. Llamada

Representa una llamada atendida.

```
Practica > JS Llamada.js > ...
1   const Operador = require('./Operador');
2   const Cliente = require('./Cliente');
3
4   class Llamada {
5       constructor(idOperador, nombreOperador, estrellasstr, idCliente, nombreCliente) {
6           this.operador = new Operador(idOperador, nombreOperador);
7           this.cliente = new Cliente(idCliente, nombreCliente);
8           this.estrellas = estrellasstr.split(';').filter(e => e.trim() === 'x').length;
9       }
10
11      clasificacion() {
12          if (this.estrellas >= 4) return "Buena";
13          if (this.estrellas >= 2) return "Media";
14          return "Mala";
15      }
16  }
17
18  module.exports = Llamada;
```

6.1.3. Operador

Representa a un operador del CallCenter.

```
Practica > JS Operador.js > Operador
1   class Operador {
2       constructor(id, nombre) {
3           this.id = id;
4           this.nombre = nombre;
5       }
6   }
7
8   module.exports = Operador;
```

6.1.4. Cliente

Representa a un cliente que recibió una llamada.

```
Practica > JS Cliente.js > Cliente
1  class Cliente {
2      constructor(id, nombre) {
3          this.id = id;
4          this.nombre = nombre;
5      }
6  }
7
8  module.exports = Cliente;
```

6.2 Métodos Principales:

METODO	DESCRIPCIÓN
cargarLlamadas(ruta)	Lee archivo.csv, procesa cada línea y crea instancias de Llamada
exportarHistorial()	Genera reportes/historial.htmlcon todas las llamadas
exportarOperadores()	Genera reportes/operadores.htmlcon lista de operadores
exportarClientes()	Genera reportes/clientes.htmlcon lista de clientes
exportarRendimiento()	Calcula % de atención por operador y genera HTML
mostrarPorcentajeClasificacion()	Muestra en consola % de llamadas buenas, medias, malas
mostrarCantidadPorEstrellas()	Muestra en consola cuántas llamadas tienen 1 a 5 estrellas

```
exportarHistorial(ruta = 'reportes/historial.html') {
    let html = this.crearEncabezadoHTML("Historial de Llamadas");
    html += `
    <h2>Historial de Llamadas</h2>
    <table border="1" cellpadding="8" cellspacing="0">
        <tr>
            <th>ID Operador</th>
            <th>Nombre Operador</th>
            <th>Estrellas</th>
            <th>ID Cliente</th>
            <th>Nombre Cliente</th>
            <th>Clasificación</th>
        </tr>`;

    this.llamadas.forEach(l => {
        const estrellasVisuales = '★'.repeat(l.estrellas) + '☆'.repeat(5 - l.estrellas);
        html += `
        <tr>
            <td>${l.operador.id}</td>
            <td>${l.operador.nombre}</td>
            <td>${estrellasVisuales}</td>
            <td>${l.cliente.id}</td>
            <td>${l.cliente.nombre}</td>
            <td>${l.clasificacion()}</td>
        </tr>`;
    });

    html += `</table></body></html>`;
    fs.writeFileSync(ruta, html);
    console.log(`Historial exportado a ${ruta}`);
}
```

```
exportarOperadores(ruta = 'reportes/operadores.html') {
    let html = this.crearEncabezadoHTML("Listado de Operadores");
    html += `<h2>Operadores</h2><table border="1" cellpadding="8"><tr><th>ID</th><th>Nombre</th></tr>`;
    for (let op of this.operadores.values()) {
        html += `<tr><td>${op.id}</td><td>${op.nombre}</td></tr>`;
    }
    html += `</table></body></html>`;
    fs.writeFileSync(ruta, html);
    console.log(`Operadores exportados a ${ruta}`);
}

exportarClientes(ruta = 'reportes/clientes.html') {
    let html = this.crearEncabezadoHTML("Listado de Clientes");
    html += `<h2>Clientes</h2><table border="1" cellpadding="8"><tr><th>ID</th><th>Nombre</th></tr>`;
    for (let cli of this.clientes.values()) {
        html += `<tr><td>${cli.id}</td><td>${cli.nombre}</td></tr>`;
    }
    html += `</table></body></html>`;
    fs.writeFileSync(ruta, html);
    console.log(`Clientes exportados a ${ruta}`);
}
```

```
exportarRendimiento(ruta = 'reportes/rendimiento.html') {
    const total = this.llamadas.length;
    const llamadasPorOp = new Map();
    this.llamadas.forEach(l => {
        llamadasPorOp.set(l.operador.id, (llamadasPorOp.get(l.operador.id) || 0) + 1);
    });

    let html = this.crearEncabezadoHTML("Rendimiento de Operadores");
    html += `<h2>Rendimiento de Operadores</h2>
            <table border="1" cellpadding="8">
            <tr><th>ID</th><th>Nombre</th><th>Porcentaje de Atención</th></tr>`;

    for (let [id, op] of this.operadores) {
        const atendidas = llamadasPorOp.get(id) || 0;
        const porcentaje = total > 0 ? ((atendidas / total) * 100).toFixed(2) : 0;
        html += `<tr><td>${op.id}</td><td>${op.nombre}</td><td>${porcentaje}%</td></tr>`;
    }

    html += `</table></body></html>`;
    fs.writeFileSync(ruta, html);
    console.log(`Rendimiento exportado a ${ruta}`);
}
```

VII. Diagramas

Diagrama de Flujo General

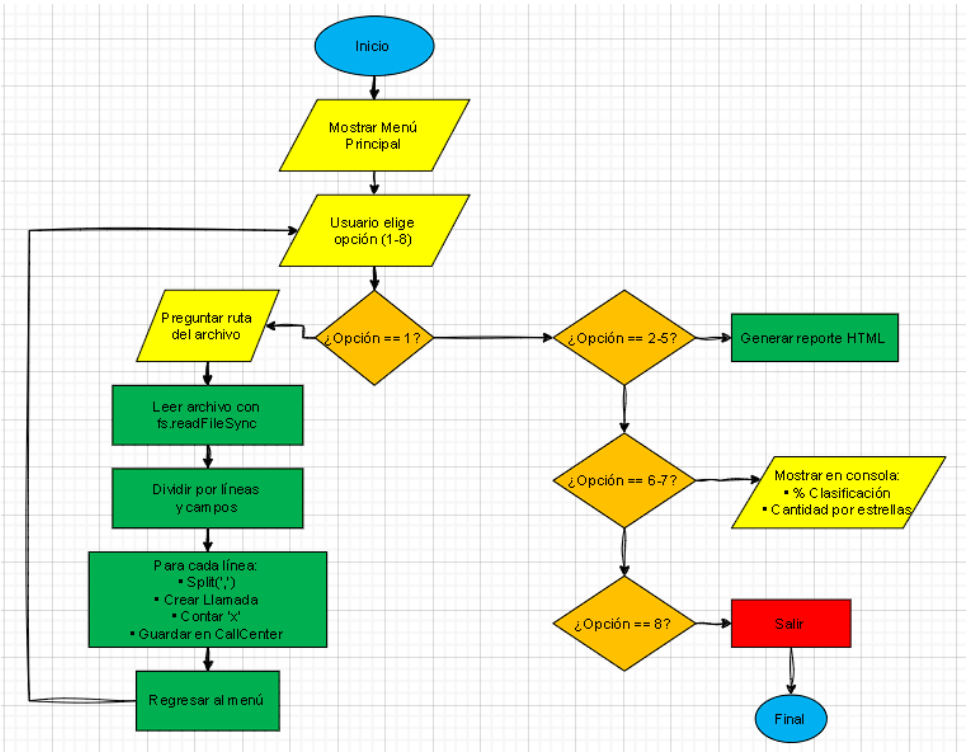
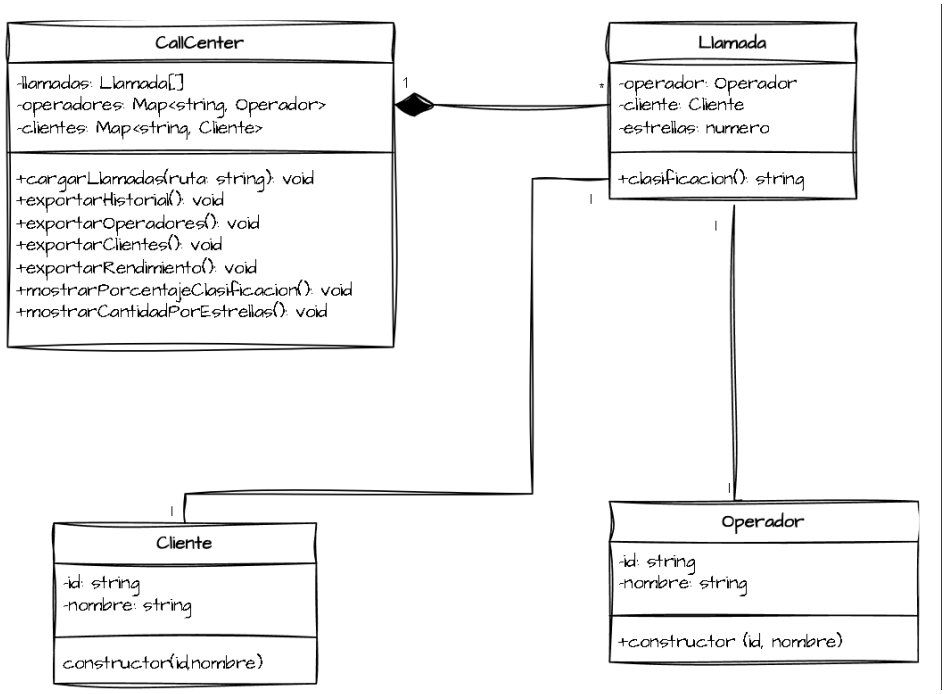


Diagrama de Clases General



VII. Conclusiones

Este Simulador de CallCenter es una implementación en JavaScript (Node.js) que permite cargar, procesar y analizar registros de llamadas de forma eficiente, sin el uso de librerías externas.
Con una estructura modular basada en POO, el sistema es fácil de mantener y extender. Permite:

- Cargar archivos .csv con formato personalizado
- Generar reportes en HTML
- Mostrar estadísticas en consola
- Cumplir con los requisitos de la práctica de LFP

La aplicación es robusta, clara y sigue buenas prácticas de programación, lo que la convierte en un ejemplo válido de uso de JavaScript para el análisis de entradas de texto.