MANUAL TÉCNICO

Práctica 1 - Simulador de Call Center

25 DE AGOSTO DE 2025

LENGUAJES FORMALES Y DE PROGRAMACIÓN
Alejandro Girón - 202405935

Introducción

Este manual trata de explicar de la mejor manera cada método que se implementó para desarrollar el programa de simulación de un Call Center.

Cada explicación contendrá el fragmento de código que representa dicho método, esto con el fin de que la práctica pueda ser replicada a quien interese o bien, alguien que empieza en el mundo de desarrollo de software usando el lenguaje de JavaScript (JS).

Arquitectura del sistema

Carpeta models

- CallsRecord.js

Carpeta Reportes

- Clientes.html
- Historial.html
- Operadores.html
- Rendimiento.html

(estos reportes fueron generados por medio de una función del archivo ReportesServicio.js)

Carpeta Services

- DocumentoServicio.js
- MenuServicio.js
- ReporteServicios.js

Carpeta Utils:

- Parseador.js

Clase principal del código:

- Index.js

Clase CallsRecord.js

Ruta: ./models/CallsRecord.js

Esta clase es un constructor el cual define la iniciación de los parámetros de entrada del archivo con extensión .csv

models > JS CallsRecord.js > CallRecord

```
export default class CallRecord{
    constructor(idOperador, NombreOperador, estrellas, idCliente, NombreCliente){
        this.idOperador = idOperador;
        this.NombreOperador = NombreOperador;
        this.estrellas = estrellas;
        this.idCliente = idCliente;
        this.NombreCliente = NombreCliente;
    }
}
```

Parseador.js

Ruta: ./utils/parseador.js

Esta clase, como su nombre lo dice, convierte (parsea) los datos de entrada de los archivos .csv dependiendo del tipo de dato. Por ejemplo, algunos datos se quedan de tipo cadena (string) como los nombres de los clientes y otros como enteros (int) así como los identificadores de los clientes

utils > JS parseador.js > parseLine

```
import CallRecord from "../models/CallsRecord.js"

export function parseLine(line) {
  const parts = line.split(",") // separando por medio de comas los datos del archivo txt

  const idOperador = Number.parseInt(parts[0]) // parseando a tipo int el id del operador
  const nombreOperador = parts[1] // separando por comas el nombre del operador
  const estrellas = parts[2].trim() // separando por comas el numero de estrellas
  const idCliente = Number.parseInt(parts[3]) // parseando a tipo int el id del cliente
  const nombreCliente = parts[4] // separando el nombre del cliente

  return new CallRecord(idOperador, nombreOperador, estrellas, idCliente, nombreCliente)
}
```

DocumentoServicios.js

Esta clase contiene solamente una función que se encarga de leer todos los archivos de tipo .csv que se intenten cargar al sistema.

```
export function LeerArchivo(filePath, callback) {
  try {
    console.log(`Leyendo archivo: ${filePath}`)

    let normalizedPath = filePath.trim()

    // quitando las comillas de la ruta absoluta
    normalizedPath = normalizedPath.replace(/^["']|["']$/g, "")

    //convirtiendo los slash "/" a la forma "\" para que windows reconozca bien la ruta y no hayan problemas
     if (process.platform === "win32") {
      normalizedPath = normalizedPath.replace(/\//g, "\\")
    }

    // verificación de ruta absoluta al momento de ingresarla a la consola
    if (!path.isAbsolute(normalizedPath)) {
      normalizedPath = path.resolve(normalizedPath)
    }

    console.log(` Ruta normalizada: ${normalizedPath}`)

    //revisando si el archivo existe y puede leerse
    fs.access(normalizedPath, fs.constants.F_OK | fs.constants.R_OK, (accessErr) => {
    //en caso de obtener un error y no se pueda acceder al archivo:
      if (accessErr) {
        console.log(` ERROR: No se puede acceder al archivo '${normalizedPath}'`)
```

```
//revisando si el archivo existe y puede leerse
fs.access(normalizedPath, fs.constants.F_OK | fs.constants.R_OK, (accessErr) => {
//en caso de obtener un error y no se pueda acceder al archivo:
  if (accessErr) {
    console.log(` ERROR: No se puede acceder al archivo '${normalizedPath}'`)


    //si obtenemos un error al no poder entrar al archivo:
    if (accessErr.code === "ENOENT") {
      console.log(` 💡 El archivo no existe en la ruta especificada`)
    }

    //retornando lista vacía "callback" cuando no se pueda leer el archivo
    if (callback) callback([])
    return
  }

  //leyendo el archivo...
  fs.readFile(normalizedPath, "utf8", (readErr, contenido) => {
    //en caso de un error de lectura...
    if (readErr) {
      console.log(` Error al leer el archivo: ${readErr.message}`)
    //retornando lista vacía para que no se rompa el programa
      if (callback) callback([])
      return
    }
```

```
  try {
    const lines = contenido
  .replace(/\r\n?|\n/g, "\n")
  .split("\n")
  .map(line => line.trim())
  .filter(line => line.length > 0 && !line.startsWith("#"))

  //si el archivo está vacío notificarlo y retornar lista vacía para mantener una buena ejecución
    if (lines.length === 0) {
      console.log(" El archivo está vacío o no contiene datos válidos")
      if (callback) callback([])
      return
    }

    console.log(`Procesando ${lines.length} líneas del archivo CSV...`)

    const validRecords = [] //contenedor para los registros correctos
    let invalidLines = 0 //contenedor para registros malos

    lines.forEach((line, index) => {
      try {
        const parts = line.split(",").map((part) => {

          return part.trim().replace(/^["']|["']$/g, "")
        })
```

```
  //ignorando lineas extras que no entran en el formato esperado
    if (parts.length !== 5) {
      console.log(` Línea ${index + 1} ignorada: formato incorrecto (${parts.length} campos en lugar de 5)`)
      invalidLines++
      return
    }

    const record = parseLine(line)
    if (record) {
      validRecords.push(record)
    }
  } catch (error) {
    console.log(` Línea ${index + 1} ignorada: ${error.message}`)
    invalidLines++
  }
})

//condicional de que si las lineas válidas es igual a 0
if (validRecords.length === 0) {
  console.log(" No se pudieron procesar registros válidos del archivo")
  console.log(" Formato requerido: id_operador,nombre_operador,estrellas,id_cliente,nombre_cliente")
  console.log(" Ejemplo: 1,Carlos Salazar,xxxxx,101,Juan Perez")
  if (callback) callback([])
  return
}
```

```
            return
          }
          //confirmando carga de archivos
          console.log(
            ` Se cargaron ${validRecords.length} registros correctamente desde ${path.basename(normalizedPath)}`,
          )
          //validación de lineas incorrectas
          if (invalidLines > 0) {
            console.log(` ${invalidLines} líneas fueron ignoradas por formato incorrecto`)
          }
          console.log("==== Archivo CSV cargado exitosamente ====")

          if (callback) callback(validRecords)
        } catch (error) {
          console.log(" Error al procesar el contenido del archivo: ", error.message)
          if (callback) callback([])
        }
      })
    })
  } catch (error) {
    console.log(" Error inesperado al procesar el archivo: ", error.message)
    console.log(" Formato requerido del archivo CSV:")
    console.log("  id_operador,nombre_operador,estrellas,id_cliente,nombre_cliente")
    console.log("  Ejemplo: 1,Carlos Salazar,xxxxx,101,Juan Perez")

    if (callback) callback([])
    return []
  }
}
```

MenuServicios.js

Ruta: ./services/MenuServicios.js

Esta clase contiene la función con el menú de opciones con las cuales podemos interactuar con el programa

```
services > JS MenuServicios.js > startMenu > handleMenuOption
import readline from "readline"
import { LeerArchivo } from "./DocumentoServicios.js"
import {exportarHistorialHTML,exportarOperadoresHTML,exportarClientesHTML,exportarRendimientoHTML,mostrarPorcentajeClasificacion,
  mostrarCantidadPorCalificacion,
} from "./ReporteServicios.js"

let records = []

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

  function showMenu() {
    console.log("\n=== MENÚ PRINCIPAL ===")
    console.log("1. Cargar Registros de Llamadas")
    console.log("2. Exportar Historial de Llamadas")
    console.log("3. Exportar Listado de Operadores")
    console.log("4. Exportar Listado de Clientes")
    console.log("5. Exportar Rendimiento de Operadores")
    console.log("6. Mostrar Porcentaje de Clasificación de Llamadas")
    console.log("7. Mostrar Cantidad de Llamadas por Calificación")
    console.log("8. Salir")
    console.log("======================")
    rl.question("Seleccione una opción: ", handleMenuOption)
  }
```

```
function handleMenuOption(option) {
  switch (option) {
    case "1":
      rl.question("Ingrese la ruta del archivo (TXT o CSV): ", (filePath) => {
        if (!filePath.trim()) {
          filePath = "./data/llamadas.txt" // Default file
          console.log(" Usando archivo por defecto: ./data/llamadas.txt")
        }

        // Validate file extension
        const validExtensions = [".txt", ".csv"]
        const fileExtension = filePath.toLowerCase().substring(filePath.lastIndexOf("."))

        if (!validExtensions.includes(fileExtension)) {
          console.log(" Formato de archivo no válido. Use archivos .txt o .csv")
          showMenu()
          return
        }

        LeerArchivo(filePath, (loadedRecords) => {
          records = loadedRecords
          if (records.length > 0) {
            console.log(`✓ Se cargaron ${records.length} registros correctamente desde ${filePath}`)
          }
          showMenu()
        })
      })
```

```
      return
    case "2":
      if (records.length === 0) {
        console.log(" Primero debe cargar los registros de llamadas")
      } else {
        exportarHistorialHTML(records)
      }
      break
    case "3":
      if (records.length === 0) {
        console.log(" Primero debe cargar los registros de llamadas")
      } else {
        exportarOperadoresHTML(records)
      }
      break
    case "4":
      if (records.length === 0) {
        console.log(" Primero debe cargar los registros de llamadas")
      } else {
        exportarClientesHTML(records)
      }
      break
    case "5":
      if (records.length === 0) {
        console.log(" Primero debe cargar los registros de llamadas")
      } else {
        exportarRendimientoHTML(records)
      }
      break
```

```
        break
      case "6":
        if (records.length === 0) {
          console.log(" Primero debe cargar los registros de llamadas")
        } else {
          mostrarPorcentajeClasificacion(records)
        }
        break
      case "7":
        if (records.length === 0) {
          console.log("Primero debe cargar los registros de llamadas")
        } else {
          mostrarCantidadPorCalificacion(records)
        }
        break
      case "8":
        console.log("Saliendo del programa...")
        rl.close()
        return
      default:
        console.log(" Opción no válida. Intente de nuevo.")
    }
    showMenu()
  }

  showMenu()
}
```

ReporteServicios.js

Ruta: ./services/ReporteServicios.js

Este archivo se encarga de generar los reportes de diferentes servicios en maquetación (html) por medio de funciones definidas, es decir, genera el código y la tabla con dichos datos por medio de una función.


```
import fs from "fs"

// Función para contar estrellas desde el string de estrellas
function contarEstrellas(estrellas) {
  return (estrellas.match(/x/g) || []).length
}

// Función para clasificar llamadas
function clasificarLlamada(numEstrellas) {
  if (numEstrellas >= 4) return "Buena"
  if (numEstrellas >= 2) return "Media"
  return "Mala"
}

export function mostrarHistorialConsola(records) {
  console.log("\n=== HISTORIAL DE LLAMADAS ===")
  console.log("ID Op. | Nombre Operador    | Estrellas | ID Cli. | Nombre Cliente")
  console.log("-------|--------------------|-----------|---------|----------------")

  records.forEach((r) => {
    const numEstrellas = contarEstrellas(r.estrellas)
    console.log(
      `${r.idOperador.toString().padEnd(6)} | ${r.NombreOperador.padEnd(17)} | ${numEstrellas} estrellas | ${r.idCliente.toString().padEnd(7)} |
    )
  })
}
```

```
export function exportarHistorialHTML(records) {
  let html = `
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Historial de Llamadas</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        h1 { text-align: center; }
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #000; padding: 8px; text-align: left; }
        th { background-color: #ccc; }
    </style>
</head>
<body>
    <h1>Historial de Llamadas</h1>
    <table>
        <tr>
            <th>ID Operador</th>
            <th>Nombre Operador</th>
            <th>Calificación</th>
            <th>ID Cliente</th>
            <th>Nombre Cliente</th>
            <th>Clasificación</th>
        </tr>`

  records.forEach((r) => {
    const numEstrellas = contarEstrellas(r.estrellas)
    const clasificacion = clasificarLlamada(numEstrellas)
```

```
  records.forEach((r) => {
    const numEstrellas = contarEstrellas(r.estrellas)
    const clasificacion = clasificarLlamada(numEstrellas)

    html += `
        <tr>
            <td>${r.idOperador}</td>
            <td>${r.NombreOperador}</td>
            <td>${numEstrellas} estrellas</td>
            <td>${r.idCliente}</td>
            <td>${r.NombreCliente}</td>
            <td>${clasificacion}</td>
        </tr>`
  })

  html += `
    </table>
    <p>Total de llamadas: ${records.length}</p>
</body>
</html>`

  if (!fs.existsSync("./reportes")) {
    fs.mkdirSync("./reportes")
  }

  fs.writeFileSync("./reportes/historial.html", html)
  console.log("✓ Reporte HTML generado en ./reportes/historial.html")
}

export function exportarOperadoresHTML(records) {
  // Obtener operadores únicos
  const operadores = []
  const operadoresMap = new Map()

  records.forEach((r) => {
    if (!operadoresMap.has(r.idOperador)) {
      operadoresMap.set(r.idOperador, r.NombreOperador)
      operadores.push({ id: r.idOperador, nombre: r.NombreOperador })
    }
  })

  let html = `
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Listado de Operadores</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        h1 { text-align: center; }
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #000; padding: 8px; text-align: left; }
        th { background-color: #ccc; }
    </style>
</head>
<body>
    <h1>Listado de Operadores</h1>
    <table>
        <tr>
            <th>ID Operador</th>
```

```
            <th>Nombre Operador</th>
        </tr>`

  operadores.forEach((op) => {
    html += `
        <tr>
            <td>${op.id}</td>
            <td>${op.nombre}</td>
        </tr>`
  })

  html += `
    </table>
    <p>Total de operadores: ${operadores.length}</p>
</body>
</html>`

  fs.writeFileSync("./reportes/operadores.html", html)
  console.log("✓ Reporte de operadores generado en ./reportes/operadores.html")
}

export function exportarClientesHTML(records) {
  // Obtener clientes únicos
  const clientes = []
  const clientesMap = new Map()
```

```
  const clientesMap = new Map()

  records.forEach((r) => {
    if (!clientesMap.has(r.idCliente)) {
      clientesMap.set(r.idCliente, r.NombreCliente)
      clientes.push({ id: r.idCliente, nombre: r.NombreCliente })
    }
  })

  let html = `
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Listado de Clientes</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        h1 { text-align: center; }
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #000; padding: 8px; text-align: left; }
        th { background-color: #ccc; }
    </style>
</head>
<body>
    <h1>Listado de Clientes</h1>
    <table>
        <tr>
            <th>ID Cliente</th>
            <th>Nombre Cliente</th>
        </tr>`

  clientes.forEach((cliente) => {
```

```
    clientes.forEach((cliente) => {
      html += `
          <tr>
              <td>${cliente.id}</td>
              <td>${cliente.nombre}</td>
          </tr>`
    })

    html += `
      </table>
      <p>Total de clientes: ${clientes.length}</p>
  </body>
  </html>`

    fs.writeFileSync("./reportes/clientes.html", html)
    console.log("✓ Reporte de clientes generado en ./reportes/clientes.html")
  }

  export function exportarRendimientoHTML(records) {
    // Calcular rendimiento por operador
    const operadores = new Map()
    const totalLlamadas = records.length

    records.forEach((r) => {
      if (operadores.has(r.idOperador)) {
        operadores.get(r.idOperador).llamadas++
      } else {
        operadores.set(r.idOperador, {
          nombre: r.NombreOperador,
          llamadas: 1,
        })
```

```
      }
    })

    let html = `
  <!DOCTYPE html>
  <html lang="es">
  <head>
      <meta charset="UTF-8">
      <title>Rendimiento de Operadores</title>
      <style>
          body { font-family: Arial, sans-serif; margin: 20px; }
          h1 { text-align: center; }
          table { border-collapse: collapse; width: 100%; }
          th, td { border: 1px solid #000; padding: 8px; text-align: left; }
          th { background-color: #ccc; }
      </style>
  </head>
  <body>
      <h1>Rendimiento de Operadores</h1>
      <table>
          <tr>
              <th>ID Operador</th>
              <th>Nombre Operador</th>
              <th>Llamadas Atendidas</th>
              <th>Porcentaje de Atención</th>
          </tr>`

    operadores.forEach((data, id) => {
      const porcentaje = ((data.llamadas / totalLlamadas) * 100).toFixed(2)
      html += `
```

```
  operadores.forEach((data, id) => {
    const porcentaje = ((data.llamadas / totalLlamadas) * 100).toFixed(2)
    html += `
        <tr>
            <td>${id}</td>
            <td>${data.nombre}</td>
            <td>${data.llamadas}</td>
            <td>${porcentaje}%</td>
        </tr>`
  })

  html += `
    </table>
    <p>Total de llamadas globales: ${totalLlamadas}</p>
</body>
</html>`

  fs.writeFileSync("./reportes/rendimiento.html", html)
  console.log("✓ Reporte de rendimiento generado en ./reportes/rendimiento.html")
}

export function mostrarPorcentajeClasificacion(records) {
  let buenas = 0,
    medias = 0,
    malas = 0

  records.forEach((r) => {
    const numEstrellas = contarEstrellas(r.estrellas)
    const clasificacion = clasificarLlamada(numEstrellas)

    if (clasificacion === "Buena") buenas++
    else if (clasificacion === "Media") medias++
    else malas++
  })

  const total = records.length
  const porcentajeBuenas = ((buenas / total) * 100).toFixed(2)
  const porcentajeMedias = ((medias / total) * 100).toFixed(2)
  const porcentajeMalas = ((malas / total) * 100).toFixed(2)

  console.log("\n=== PORCENTAJE DE CLASIFICACIÓN DE LLAMADAS ===")
  console.log(`🟢 Llamadas Buenas (4-5 estrellas): ${buenas} (${porcentajeBuenas}%)`)
  console.log(`🟡 Llamadas Medias (2-3 estrellas): ${medias} (${porcentajeMedias}%)`)
  console.log(`🔴 Llamadas Malas (0-1 estrellas): ${malas} (${porcentajeMalas}%)`)
  console.log(`📊 Total de llamadas: ${total}`)
}

export function mostrarCantidadPorCalificacion(records) {
  const calificaciones = { 1: 0, 2: 0, 3: 0, 4: 0, 5: 0 }

  records.forEach((r) => {
    const numEstrellas = contarEstrellas(r.estrellas)
    if (numEstrellas >= 1 && numEstrellas <= 5) {
      calificaciones[numEstrellas]++
    }
  })
```

```
285    console.log("\n=== CANTIDAD DE LLAMADAS POR CALIFICACIÓN ===")
286    console.log(`⭐ 1 estrella: ${calificaciones[1]} llamadas`)
287    console.log(`⭐⭐ 2 estrellas: ${calificaciones[2]} llamadas`)
288    console.log(`⭐⭐⭐ 3 estrellas: ${calificaciones[3]} llamadas`)
289    console.log(`⭐⭐⭐⭐ 4 estrellas: ${calificaciones[4]} llamadas`)
290    console.log(`⭐⭐⭐⭐⭐ 5 estrellas: ${calificaciones[5]} llamadas`)
291    console.log(`📊 Total: ${records.length} llamadas`)
292  }
```

Index.js

Clase que instancia la función para inicializar el menú, es la clase principal que ejecuta el método

```
JS index.js
1   import { startMenu } from "./services/MenuServicios.js";
2
3   console.clear() // Limpiar la consola al iniciar
4   console.log("======= Simulador de Call Center =======");
5   startMenu(); // Iniciar el menú del programa
```

Librerías Utilizadas

fs: Operaciones de sistema de archivos (nativa)

- **path:** Manipulación de rutas (nativa)

- **readline:** Interfaz de línea de comandos (nativa)

Compatibilidad

- **Node.js:** Versión 14.0.0 o superior

- **Sistemas operativos:** Windows, macOS, Linux

- **Formatos de archivo:** CSV y TXT con separación por comas