

Selvin Raúl Chuquiej Andrade
202405516
LFP B+

MANUAL TÉCNICO

Practica 1

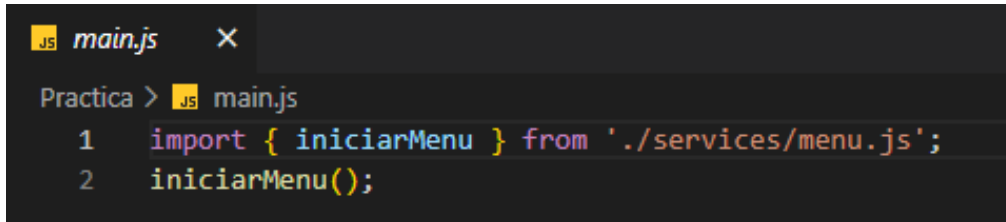
Estructura de archivos:

practica/

----- main.js	#Punto de entrada de la aplicación
----- data/	#Directorio para archivos CVS de entrada
----- reportesHTML/	#Directorio para reportes generados
----- services/	
----- menú.js	#Logica del menú interactivo
----- model/	
----- registroLlamada.js	#Modelo de datos CallRecord
----- services/	
----- fileService.js	#Servicio de manejo de archivo
----- reportes/	
----- historialLlamadas.js	#Generador historial completo
----- listadoCliente.js	#Generador listado de clientes
----- listadoOperadores.js	#Generador listado de operadores
----- rendimientoOperador.js	#Generador reporte rendimiento

- main.js:

La función principal es inicializar y controlar el menú interactivo, mostrando las opciones disponibles al usuario, validando su elección y redirigiendo el flujo hacia las funciones correspondientes según la opción seleccionada.



```
.JS main.js X
Practica > .JS main.js
1 import { iniciarMenu } from './services/menu.js';
2 iniciarMenu();
```

- services/menu.js:

iniciarMenu(): La función mostrarMenu se encarga de desplegar en consola el menú principal del sistema de registro de llamadas, imprimiendo un listado numerado de opciones que el usuario puede seleccionar. Además de presentar visualmente las alternativas disponibles (como cargar registros, exportar reportes o salir), utiliza rl.question para capturar la opción elegida por el usuario y pasarla a la función procesarOpcion, que será la responsable de manejar la lógica asociada a cada caso.

```
function mostrarMenu() {
  console.log("\n" + "=".repeat(50));
  console.log('--- Menú de Registro de Llamadas ---');
  console.log('1. Cargar registro 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("\n" + "=".repeat(50));
  rl.question('Seleccione una opción: ', procesarOpcion);
}
```

procesarOpcion(opcion): La función procesarOpcion recibe la opción ingresada por el usuario y, mediante un switch, ejecuta la acción correspondiente (cargar datos, generar reportes o mostrar estadísticas), validando cuando es necesario que haya registros cargados; tras cada acción normalmente vuelve a mostrar el menú, y en la opción '8' cierra el programa.

```
function procesarOpcion(opcion) {
  switch (opcion) {
    case '1':
      cargarRegistros();
      return;

    case '2':
      if (archivoNoCargado()) return;
      generarHistorialLlamadas(llamadas);
      mostrarMenu();
      return;

    case '3':
      if (archivoNoCargado()) return;
      generarListadoOperadores(llamadas);
      mostrarMenu();
      return;
  }
}
```

cargarRegistros(): La función cargarRegistros solicita al usuario el nombre de un archivo CSV ubicado en la carpeta data, construye la ruta correspondiente y llama a cargarArchivo para obtener los registros de llamadas; si el archivo se carga correctamente y contiene datos, muestra un mensaje de confirmación junto con el contenido, de lo contrario indica que el archivo no pudo cargarse o está vacío, y al final vuelve a mostrar el menú.

```
function cargarRegistros() {
  rl.question('\nIngresa el nombre del archivo (guardado en la carpeta data): ', (nombreArchivo) => {
    const rutaArchivo = `./data/${nombreArchivo}.csv`;
    llamadas = cargarArchivo(rutaArchivo);
    if (llamadas && llamadas.length > 0) {
      console.log('Archivo cargado correctamente.');
```

```
      console.log(llamadas);
    } else {
      console.log('No se pudo cargar el archivo o está vacío.');
```

```
    }
    mostrarMenu();
  });
}
```

porcentajeClasificacion(): La función porcentajeClasificacion analiza las llamadas cargadas y calcula el porcentaje de evaluaciones buenas, medias y malas según el número de estrellas; luego muestra en consola el porcentaje y la cantidad de llamadas de cada categoría.

```
function porcentajeClasificacion() {
  if (!llamadas || llamadas.length === 0) {
    console.log('No hay registros de llamadas cargados.');
    return;
  }

  let buenas = 0, medias = 0, malas = 0;

  llamadas.forEach(lla => {
    if (lla.no_estrellas >= 4) {
      buenas++;
    } else if (lla.no_estrellas >= 2) {
      medias++;
    } else {
      malas++;
    }
  });

  const total = llamadas.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(`Buenas (4-5 estrellas): ${porcentajeBuenas}% (${buenas} llamadas)`);
  console.log(`Medias (2-3 estrellas): ${porcentajeMedias}% (${medias} llamadas)`);
  console.log(`Malas (0-1 estrellas): ${porcentajeMalas}% (${malas} llamadas)`);
}
```

llamadasClasificacion(): La función llamadasClasificacion recorre las llamadas y cuenta cuántas tienen cada calificación de 0 a 5 estrellas, para luego mostrar en consola la cantidad de llamadas correspondientes a cada nivel de estrellas.

```
function llamadasClasificacion() {
  const contador = {
    0: 0,
    1: 0,
    2: 0,
    3: 0,
    4: 0,
    5: 0
  };

  llamadas.forEach(lla => {
    if (lla.no_estrellas >= 0 && lla.no_estrellas <= 5) {
      contador[lla.no_estrellas]++;
    }
  });

  console.log("\n--- Cantidad de Llamadas por Calificación ---");
  for (let i = 1; i <= 5; i++) {
    console.log(`${i} estrella${i > 1 ? 's' : ''}: ${contador[i]} llamadas`);
  }
}
```

archivoNoCargado(): La función archivoNoCargado valida si la variable llamadas está vacía o no existe; en ese caso, muestra un mensaje de advertencia, reinicia el menú y retorna true, indicando que no hay datos disponibles, mientras que si hay registros devuelve false.

```
function archivoNoCargado(params) {
  if (!llamadas || llamadas.length === 0) {
    console.log('No hay registros de llamadas cargados.');
    mostrarMenu();
    return true;
  }
  return false;
}
```

- **services/fileService.js**

cargarArchivo(filePath): Lee el archivo de forma síncrona haciendo que filtre líneas vacías aplicando parsing a cada línea válida y manejando errores de lectura con mensajes descriptivos

```
export function cargarArchivo(filePath) {
  try {
    const data = fs.readFileSync(filePath, 'utf8');
    const line = data.split('\n').filter(line => line.trim() !== '');
    return line.map(parseLine)
  } catch (error) {
    console.log(`Error al cargar el archivo: ${error.message}`);
  }
}
```

parseLine(line): Divide cada línea por comas procesando el campo de estrellas contando las "X" separadas por ";" y construyendo objetos CallRecord con los datos procesados, así como convirtiendo los campos numéricos a los tipos adecuados

```
function parseLine(line) {
  const parts = line.split(',');
  const estrellas = parts[2].trim().split(';');
  const noEstrellas = estrellas.filter(val => val.trim().toLowerCase() === 'x').length;
  return new CallRecord(
    parseInt(parts[0].trim()),
    parts[1].trim(),
    noEstrellas,
    parseInt(parts[3].trim()),
    parts[4].trim()
  );
}
```

- **model/registroLlamadas.js:**

Define la estructura de datos fundamental del sistema

```
export default class CallRecord {
  constructor(id_operador, nombre_operador, no_estrellas, id_cliente, nombre_cliente) {
    this.id_operador = id_operador;
    this.nombre_operador = nombre_operador;
    this.no_estrellas = no_estrellas;
    this.id_cliente = id_cliente;
    this.nombre_cliente = nombre_cliente;
  }
}
```

- **reportes/historialLlamadas.js:**

La función generarHistorialLlamadas crea un archivo HTML que contiene una tabla con los datos de todas las llamadas proporcionadas. Recorre el arreglo llamadas, construye dinámicamente el contenido de la tabla con los detalles de operadores y clientes, elimina cualquier archivo previo con el mismo nombre y guarda el nuevo reporte en ./reportesHTML/historialLlamadas.html, mostrando un mensaje de confirmación al finalizar.

```

export function generarHistorialLlamadas(llamadas) {
  let html = `
    <!DOCTYPE html>
    <html lang="es">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Historial de Llamadas</title>
      <style>
        body { font-family: Arial, sans-serif; text-align: center; }
        table { width: 75%; border-collapse: collapse; margin-left: auto; margin-right: auto; }
        th, td { border: 1px solid #000000ff; padding: 8px; text-align: left; }
        th { background-color: #ffffff; }
      </style>
    </head>
    <body>
      <h1>Historial de Llamadas</h1>
      <table>
        <thead>
          <tr>
            <th>Id Operador</th>
            <th>Nombre Operador</th>
            <th>Número de Estrellas</th>
            <th>Id Cliente</th>
            <th>Nombre Cliente</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td>${lla.id_operador}</td>
            <td>${lla.nombre_operador}</td>
            <td>${lla.no_estrellas}</td>
            <td>${lla.id_cliente}</td>
            <td>${lla.nombre_cliente}</td>
          </tr>
        </tbody>
      </table>
    </body>
  </html>`;

  if (fs.existsSync('./reportesHTML/historialLlamadas.html')) {
    fs.unlinkSync('./reportesHTML/historialLlamadas.html');
  }

  fs.writeFileSync('./reportesHTML/historialLlamadas.html', html, 'utf8');
  console.log('Reporte generado: ./reportesHTML/historialLlamadas.html');
}

```

- reportes/listadoCliente.js:

La función generarListadoClientes crea un archivo HTML con una tabla que lista todos los clientes únicos de las llamadas proporcionadas. Primero construye un mapa para evitar duplicados por id_cliente, luego genera dinámicamente el contenido de la tabla, elimina cualquier archivo previo con el mismo nombre y guarda el nuevo reporte en ./reportesHTML/listadoClientes.html, mostrando un mensaje de confirmación al finalizar.

```

export function generarListadoClientes(llamadas) {
  const clientesMap = {};

  llamadas.forEach(lla => {
    const idCl = lla.id_cliente;
    const nombreCl = lla.nombre_cliente;
    if (!clientesMap[idCl]) {
      clientesMap[idCl] = {
        id_cliente: idCl,
        nombre_cliente: nombreCl
      };
    }
  });

  const clientes = Object.values(clientesMap);

  let html = `
    <!DOCTYPE html>
    <html lang="es">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Listado de Clientes</title>
      <style>
        body { font-family: Arial, sans-serif; text-align: center; }
        table { width: 45%; border-collapse: collapse; margin-left: auto; margin-right: auto; }
        th, td { border: 1px solid #000000ff; padding: 8px; text-align: left; }
        th { background-color: #ffffff; }
      </style>
    </head>
    <body>
      <h1>Listado de Clientes</h1>
      <table>
        <thead>
          <tr>
            <th>Id Cliente</th>
            <th>Nombre Cliente</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td>${cl.id_cliente}</td>
            <td>${cl.nombre_cliente}</td>
          </tr>
        </tbody>
      </table>
    </body>
  </html>`;

  if (fs.existsSync('./reportesHTML/listadoClientes.html')) {
    fs.unlinkSync('./reportesHTML/listadoClientes.html');
  }

  fs.writeFileSync('./reportesHTML/listadoClientes.html', html, 'utf8');
  console.log('Reporte generado: ./reportesHTML/listadoClientes.html');
}

```

```

html += `
  </tbody>
</table>
</body>
</html>`;

if (fs.existsSync('./reportesHTML/listadoClientes.html')) {
  fs.unlinkSync('./reportesHTML/listadoClientes.html');
}

fs.writeFileSync('./reportesHTML/listadoClientes.html', html, 'utf8');
console.log('Reporte generado: ./reportesHTML/listadoClientes.html');
}

```

- **reportes/listadoOperadores.js:**

La función generarListadoOperadores crea un archivo HTML con una tabla que lista todos los operadores únicos de las llamadas proporcionadas. Construye un mapa para evitar duplicados por id_operador, genera el contenido de la tabla, elimina cualquier archivo previo con el mismo nombre y guarda el nuevo reporte en ./reportesHTML/listadoOperadores.html, mostrando un mensaje de confirmación al finalizar.

```
export function generarListadoOperadores(llamadas) {  
  
  const operadoresMap = {};  
  
  llamadas.forEach(lla => {  
    const idOp = lla.id_operador;  
    const nombreOp = lla.nombre_operador;  
    if (!operadoresMap[idOp]) {  
      operadoresMap[idOp] = {  
        id_operador: idOp,  
        nombre_operador: nombreOp  
      };  
    }  
  });  
  
  const operadores = Object.values(operadoresMap);  
}
```

```
let html = `  
  <!DOCTYPE html>  
  <html lang="es">  
  <head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Listado de Operadores</title>  
    <style>  
      body { font-family: Arial, sans-serif; text-align: center; }  
      table { width: 45%; border-collapse: collapse; margin-left: auto; margin-right: auto; }  
      th, td { border: 1px solid #000000ff; padding: 8px; text-align: left; }  
      th { background-color: #ffffff; }  
    </style>  
  </head>  
  <body>  
    <h1>Listado de Operadores</h1>  
    <table>  
      <thead>  
        <tr>  
          <th>Id Operador</th>  
          <th>Nombre Operador</th>  
        </tr>  
      </thead>  
      <tbody>`;
```

```
operadores.forEach(op => {  
  html += `  
    <tr>  
      <td>${op.id_operador}</td>  
      <td>${op.nombre_operador}</td>  
    </tr>`;  
});  
  
html += `  
  </tbody>  
</table>  
</body>  
</html>`;  
  
if (fs.existsSync('./reportesHTML/listadoOperadores.html')) {  
  fs.unlinkSync('./reportesHTML/listadoOperadores.html');  
}  
  
fs.writeFileSync('./reportesHTML/listadoOperadores.html', html, 'utf8');  
console.log('Reporte generado: ./reportesHTML/listadoOperadores.html');
```

- **reportes/rendimientoOperadores.js:**

La función `generarRendimientoOperadores` crea un archivo HTML que muestra el rendimiento de cada operador, calculando el total de llamadas atendidas y su porcentaje respecto al total de llamadas. Primero construye un mapa para agrupar las llamadas por operador, calcula el porcentaje de atención, genera la tabla en HTML, elimina cualquier archivo previo con el mismo nombre y guarda el reporte en `./reportesHTML/rendimientoOperadores.html`, mostrando un mensaje de confirmación al finalizar.

```
export function generarRendimientoOperadores(llamadas) {  
  
  const operadoresMap = {};  
  
  llamadas.forEach(ll => {  
    const idOp = ll.id_operador;  
    const nombreOp = ll.nombre_operador;  
  
    if (!operadoresMap[idOp]) {  
      operadoresMap[idOp] = {  
        id_operador: idOp,  
        nombre_operador: nombreOp,  
        totalLlamadas: 0  
      };  
    }  
  
    operadoresMap[idOp].totalLlamadas++;  
  });  
  
  const operadores = Object.values(operadoresMap);  
  const totalLlamadas = llamadas.length;  
  
  operadores.forEach(op => {  
    op.porcentaje_atencion = ((op.totalLlamadas / totalLlamadas) * 100).toFixed(2) + '%';  
  });  
  
  let html = `  
    <!DOCTYPE html>  
    <html lang="es">  
      <head>  
        <meta charset="UTF-8">  
        <meta name="viewport" content="width=device-width, initial-scale=1.0">  
        <title>Rendimiento de Operadores</title>  
        <style>  
          body { font-family: Arial, sans-serif; text-align: center; }  
          table { width: 45%; border-collapse: collapse; margin-left: auto; margin-right: auto; }  
          th, td { border: 1px solid #000000ff; padding: 8px; text-align: left; }  
          th { background-color: #ffffff; }  
        </style>  
      </head>  
      <body>  
        <h1>Rendimiento de Operadores</h1>  
        <table>  
          <thead>  
            <tr>  
              <th>Id Operador</th>  
              <th>Nombre Operador</th>  
              <th>Total Llamadas</th>  
              <th>Porcentaje de atencion</th>  
            </tr>  
          </thead>  
          <tbody>  
            <tr>  
              <td>${op.id_operador}</td>  
              <td>${op.nombre_operador}</td>  
              <td>${op.totalLlamadas}</td>  
              <td>${op.porcentaje_atencion}</td>  
            </tr>  
          </tbody>  
        </table>  
      </body>  
</html>`;  
  
  operadores.forEach(op => {  
    html += `            <tr>  
              <td>${op.id_operador}</td>  
              <td>${op.nombre_operador}</td>  
              <td>${op.totalLlamadas}</td>  
              <td>${op.porcentaje_atencion}</td>  
            </tr>`;  
  });  
  
  html += `          </tbody>  
        </table>  
      </body>  
</html>`;  
  
  if (fs.existsSync('./reportesHTML/rendimientoOperadores.html')) {  
    fs.unlinkSync('./reportesHTML/rendimientoOperadores.html');  
  }  
  
  fs.writeFileSync('./reportesHTML/rendimientoOperadores.html', html, 'utf8');  
  console.log('Reporte generado: ./reportesHTML/rendimientoOperadores.html');  
}
```