Manual Técnico

CARLOS DÍAZ - 202400245

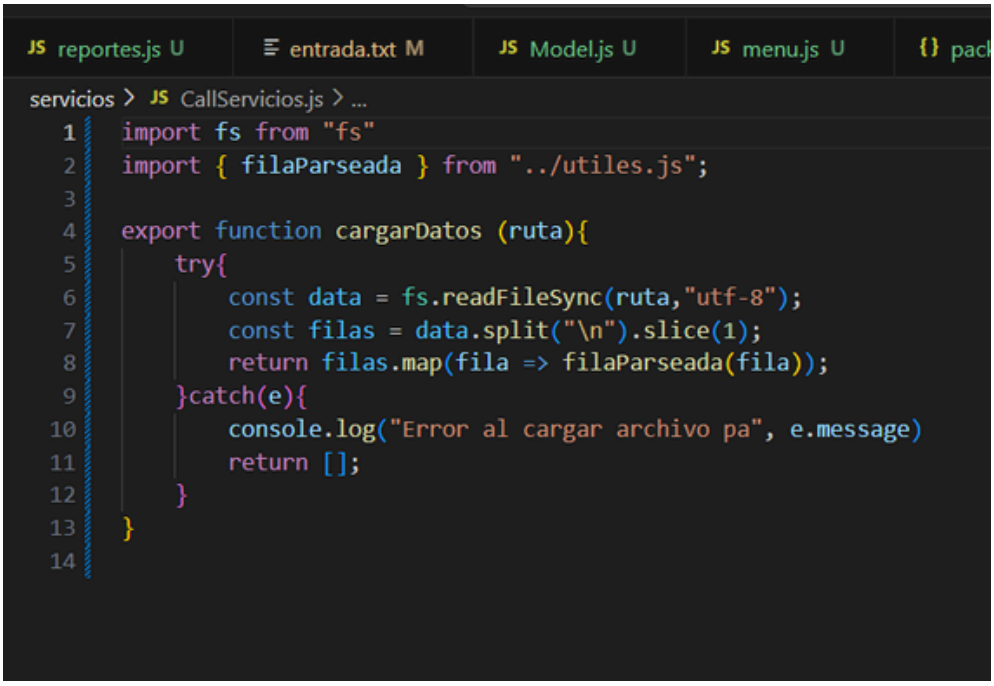
Menú

```
function mostrarMenu (){
    console.log("\n--------------- Menú-------------");
    console.log("1 => cargar datos");
    console.log("2 => Mostrar registros en consola");
    console.log("3 => generar pagina web");
    console.log("4 => salir");
    rl.question("Seleccione una opcion: \n",seleccion);
    ;
}
```

Tenemos nuestro Menú sencillo de consola que después con un callback llamamos selección que es una función de nuestro switch case, en el primer case tenemos la ruta del archivo y su nombre, en los demas mandamos a llamar funciones exportadas.

```
function seleccion(opcion){
    switch (opcion) {
        case "1":
                llamadas = cargarDatos("./ArchivosDeDatos/entrada.txt")
                    if(llamadas){
                        console.log("Se cargo archivo");
                        mostrarMenu();
                    }else{
                        console.log("sucedio un fallo al cargar archivo");
                        mostrarMenu();
                    }
            break;

        case "2":
            if (llamadas != null) {

                CantidadLamadasPorClasificacion(llamadas)
                mostrarClasificacionLLamadas(llamadas);
                mostrarMenu();
            }else {
                console.log("No has cargado los datos de las llamadas");
                mostrarMenu();
            }

            break;
        case "3":
            if (llamadas != null ) {
                generarReportesHTML(llamadas)
                mostrarMenu();

            }else{
                console.log("No has cargado los datos de las llamadas");
                mostrarMenu();
```

Cargar Archivo



Para cargar el archivo tenemos este método que nos ayuda en la lectura del archivo y hacemos uso de fs, utilizamos el try catch por si llegase a ocurrir algún problema y que no nos crashee el programa, la data que extraigamos la partimos por lineas y hacemos omiso a la primer entrada que es nuestro encabezado para despúes mapear las filas y las vamos parseando con el siguiente método

Organizar filas

```
import {Model} from "./Modelos/Model.js"

export function filaParseada(fila){
    const partes = fila.split(",");
    return new Model(
        parseInt(partes[0]),
        partes[1],
        partes[2],
        parseInt(partes[3]),
        partes[4]
    )

}
```

Por cada fila que le enviamos del método anterior creamos una llamada (modelo), con la cual le vamos mandando los datos que vamos partiendo, solo tenemos que conocer como vienen los datos del txt

```
export class Model{
    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
    }

}
```

(nuestro modelo)

Reportes

Calcular Estadisticas:

Nuestro Método auxiliar con el cual calcularemos los datos que necesitamos para nuestros reportes

```
function calcularEstadisticas(llamadas) {
 // Filtrar solo llamadas válidas
 const llamadasValidas = llamadas.filter(llamada =>
     llamada &&
     llamada.id_operador &&
     llamada.nombre_operador &&
     llamada.estrellas &&
     llamada.id_cliente &&
     llamada.nombre_cliente
 );

 const totalLlamadas = llamadasValidas.length;
 const clasificacion = { Buena: 0, Media: 0, Mala: 0 };
 const porEstrella = { 0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0 };
 const operadores = {};
 const clientes = {};

 llamadasValidas.forEach(llamada => {
     const estrellas = contarEstrellas(llamada.estrellas);

     // Asegurarnos que el conteo esté en el rango 0-5
     const estrellasValidas = Math.max(0, Math.min(5, estrellas));
     porEstrella[estrellasValidas]++;

     // Clasificación
     if (estrellasValidas >= 4) clasificacion.Buena++;
     else if (estrellasValidas >= 2) clasificacion.Media++;
     else clasificacion.Mala++;

     // Operadores
     if (!operadores[llamada.id_operador]) {
         operadores[llamada.id_operador] = {
             nombre: llamada.nombre_operador,
             llamadas: 0
         };
     }
```

Primero nos aseguramos que no vengan datos nulos del archivo, para despúes crear las constantes de nuestras llamadas que usaremos para devolver las estadisticas que saquemos

```
llamadasValidas.forEach(llamada => {
    const estrellas = contarEstrellas(llamada.estrellas);

    // Asegurarnos que el conteo esté en el rango 0-5
    const estrellasValidas = Math.max(0, Math.min(5, estrellas));
    porEstrella[estrellasValidas]++;

    // Clasificación
    if (estrellasValidas >= 4) clasificacion.Buena++;
    else if (estrellasValidas >= 2) clasificacion.Media++;
    else clasificacion.Mala++;

    // Operadores
    if (!operadores[llamada.id_operador]) {
        operadores[llamada.id_operador] = {
            nombre: llamada.nombre_operador,
            llamadas: 0
        };
    }
    operadores[llamada.id_operador].llamadas++;

    // Clientes
    if (!clientes[llamada.id_cliente]) {
        clientes[llamada.id_cliente] = llamada.nombre_cliente;
    }
});
```

Recibimos un array de llamadas, entonces extraemos los datos que necesitamos por cada llamada como el conteo de las llamadas, el conteo de llamadas malas y buenas y la cantidad de llamadas que reciben nuestros operadores, después calculamos el rendimiento de los operadores con el chivo que nos dieron en el enunciado

```
const rendimiento = Object.keys(operadores).map(id => ({
    id,
    nombre: operadores[id].nombre,
    porcentaje: ((operadores[id].llamadas / totalLlamadas) * 100).toFixed(2)
}));

return {
    clasificacion,
    porEstrella,
    operadores: Object.keys(operadores).map(id => ({ id, nombre: operadores[id].nombre })),
    clientes: Object.keys(clientes).map(id => ({ id, nombre: clientes[id] })),
    rendimiento,
    totalLlamadas
};
```

Por ultimo devolvemos un objeto con los datos que extraimos del archivo para darle uso.

Conteo de estrellas:

Tenemos un método auxiliar el cual cuenta las estrellas y nos las devuelve en un entero para usarlo en nuestras estadisticas, nos aseguramos que esa parte del archivo no venga nulo y si es el caso nos devuelva 0 para después dividirlas por los “;” y despues que me devuelva el entero de estrellas.

```
function contarEstrellas(estrellasStr) {
    if (!estrellasStr || typeof estrellasStr !== 'string') return 0;

    const valores = estrellasStr.split(';')
        .map(e => e.trim())
        .filter(e => e !== '');

    return valores.slice(0, 5).filter(e => e.toLowerCase() === 'x').length;
}
```

Cantidad de llamadas por clasificación:

Haciendo uso de nuestro calculo de estadísticas mandamos a llamar este método para poder imprimir en consola las llamadas por clasificación.

```
export function CantidadLamadasPorClasificacion(llamadas) {
    const stats = calcularEstadisticas(llamadas);
    console.log("-----------Cantidad de llamadas por estrellas---------");
    for (let estrellas = 0; estrellas <= 5; estrellas++) {
        console.log(`${estrellas} estrellas: ${stats.porEstrella[estrellas]} llamadas`);
    }
}
```

Creación de HTML

```
export function generarReportesHTML(llamadas){
    const stats = calcularEstadisticas(llamadas)
    let html = `
        <!DOCTYPE html>
        <html lang="es">
        <head>
            <meta charset="UTF-8">
            <title>Historial de Llamadas</title>
            <style>
                body {
        font-family: "Segoe UI", Tahoma, sans-serif;
        background-color: #fafafa;
        color: #333;
        padding: 20px;
    }

    table {
        border-collapse: collapse;
        width: 100%;
        margin: 15px 0;
        font-size: 14px;
        box-shadow: 0 2px 6px rgba(0,0,0,0.1);
        border-radius: 6px;
        overflow: hidden;
    }

    th, td {
        padding: 10px 12px;
        text-align: left;
    }
```

Con un poco de css definimos las tablas que meteremos para que no se vea tan mal

```
<body>
    <h1>Reporte Completo del CallCenter</h1>

    <!-- Sección de Estadísticas Generales -->
    <h2>Estadísticas Generales</h2>
    <div class="stats-container">
        <div class="stat-card">
            <h3>Total de Llamadas</h3>
            <div class="stat-value">${stats.totalLlamadas}</div>
        </div>
        <div class="stat-card">
            <h3>Llamadas Buenas</h3>
            <div class="stat-value">${stats.clasificacion.Buena}</div>
            <div>(${((stats.clasificacion.Buena / stats.totalLlamadas) * 100).toFixed(2)}%)</div>
        </div>
        <div class="stat-card">
            <h3>Llamadas Medias</h3>
            <div class="stat-value">${stats.clasificacion.Media}</div>
            <div>(${((stats.clasificacion.Media / stats.totalLlamadas) * 100).toFixed(2)}%)</div>
        </div>
        <div class="stat-card">
            <h3>Llamadas Malas</h3>
            <div class="stat-value">${stats.clasificacion.Mala}</div>
            <div>(${((stats.clasificacion.Mala / stats.totalLlamadas) * 100).toFixed(2)}%)</div>
        </div>
    </div>
```

Creamos la primer tabla que contiene el registro completo de nuestro call center

```
// Historial de llamadas
llamadas.forEach(llamada => {
    html += `
            <tr>
                <td>${llamada.id_operador}</td>
                <td>${llamada.nombre_operador}</td>
                <td>${llamada.estrellas} (${contarEstrellas(llamada.estrellas)} estrellas)</td>
                <td>${llamada.id_cliente}</td>
                <td>${llamada.nombre_cliente}</td>
            </tr>
    `;
});

// Sección de Operadores
html += `
        </table>

        <h2>Listado de Operadores</h2>
        <table>
            <tr>
                <th>ID</th>
                <th>Nombre</th>
            </tr>
`;
stats.operadores.forEach(op => {
    html += `
            <tr>
                <td>${op.id}</td>
                <td>${op.nombre}</td>
            </tr>
    `;
```

Luego para cada registro de llamadas filtramos lo que nuestro método de estadísticas nos devuelve para realizar un registro en la tabla, luego realizamos en html guardandolo en la ruta

```
// Guardar el archivo
if (!fs.existsSync("./Reportes")) {
    fs.mkdirSync("./Reportes");
}
fs.writeFileSync("./Reportes/reporte_completo.html", html);
console.log("Reporte generado exitosamente en ./Reportes/reporte_completo.html");
```

Primero nos aseguramos que exista la ruta y guardamos ahí el html para después verlo en algún navegador siempre de manera local

Diagrama de clases

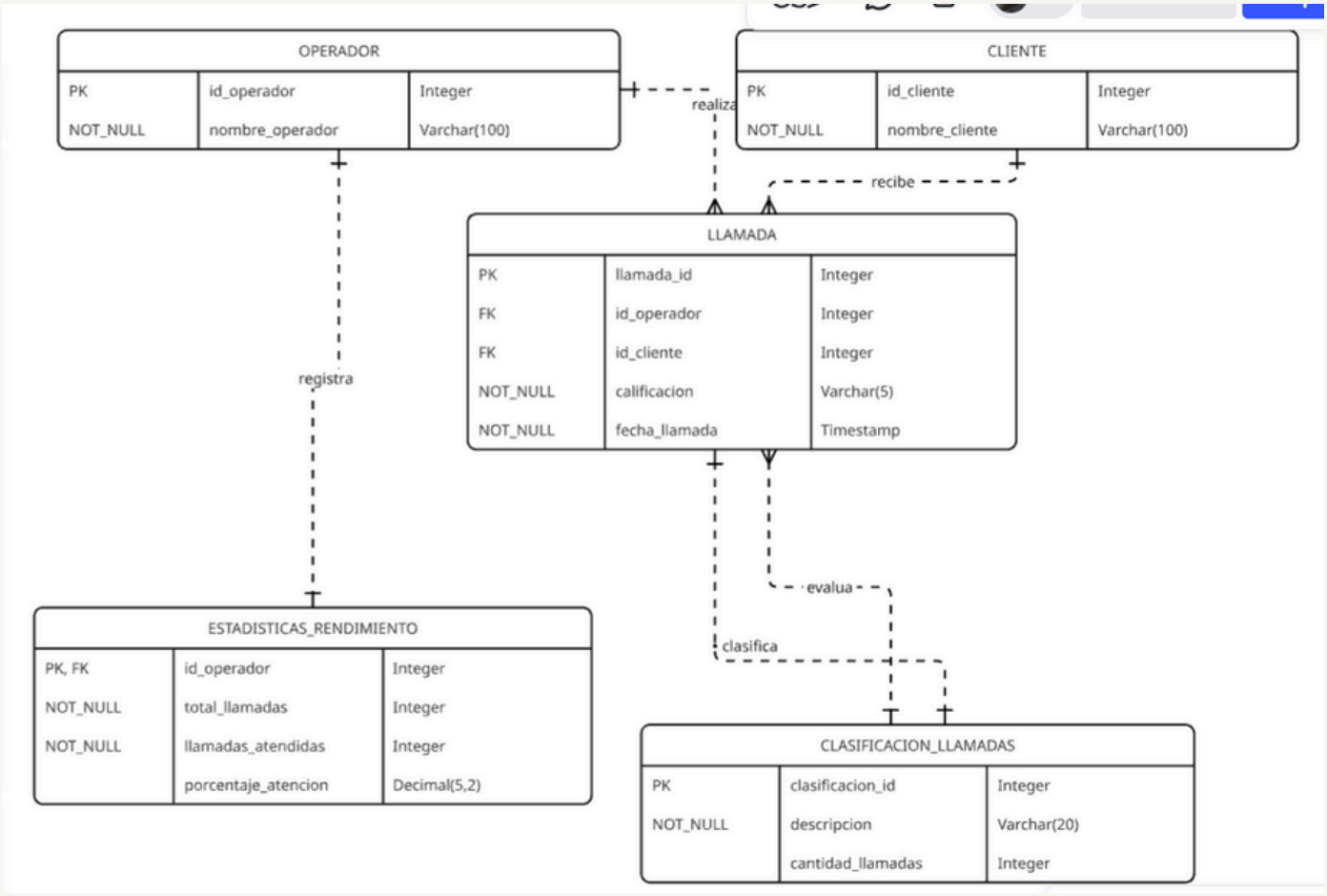
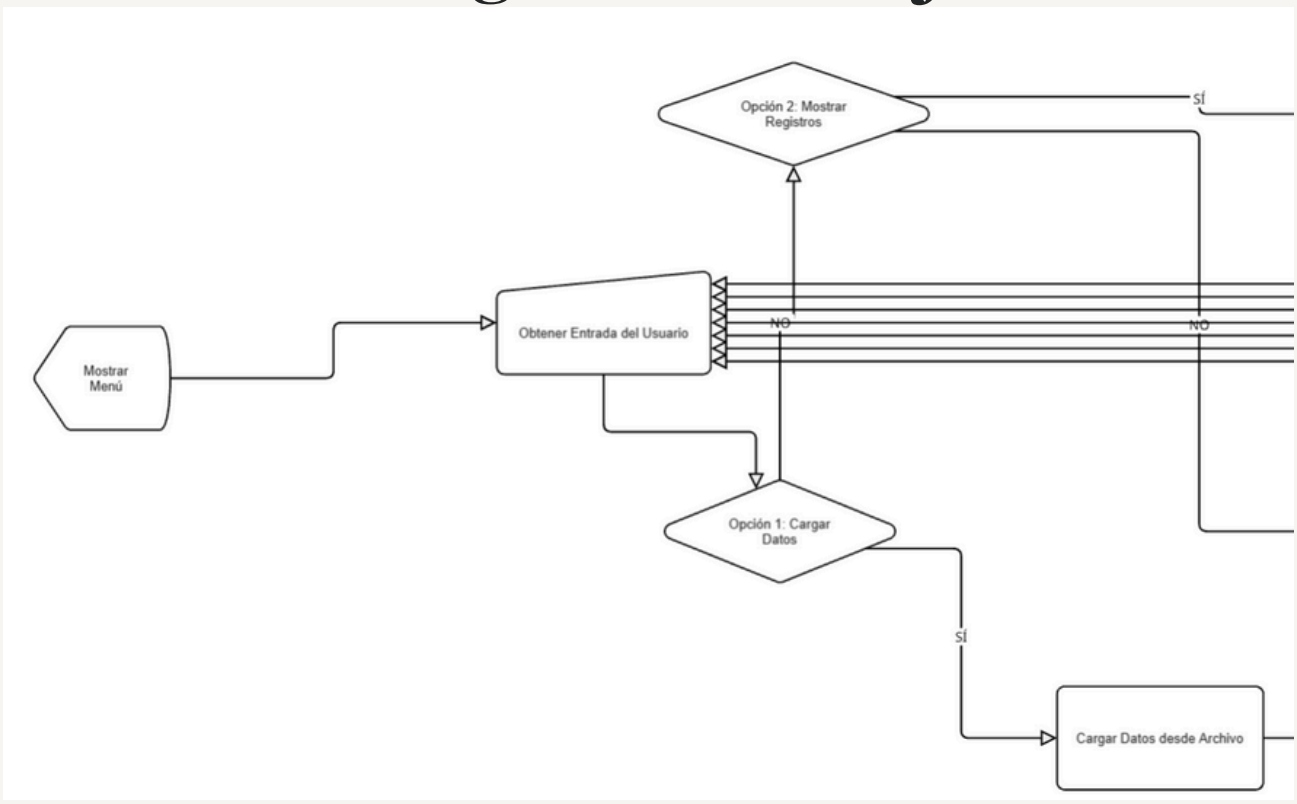
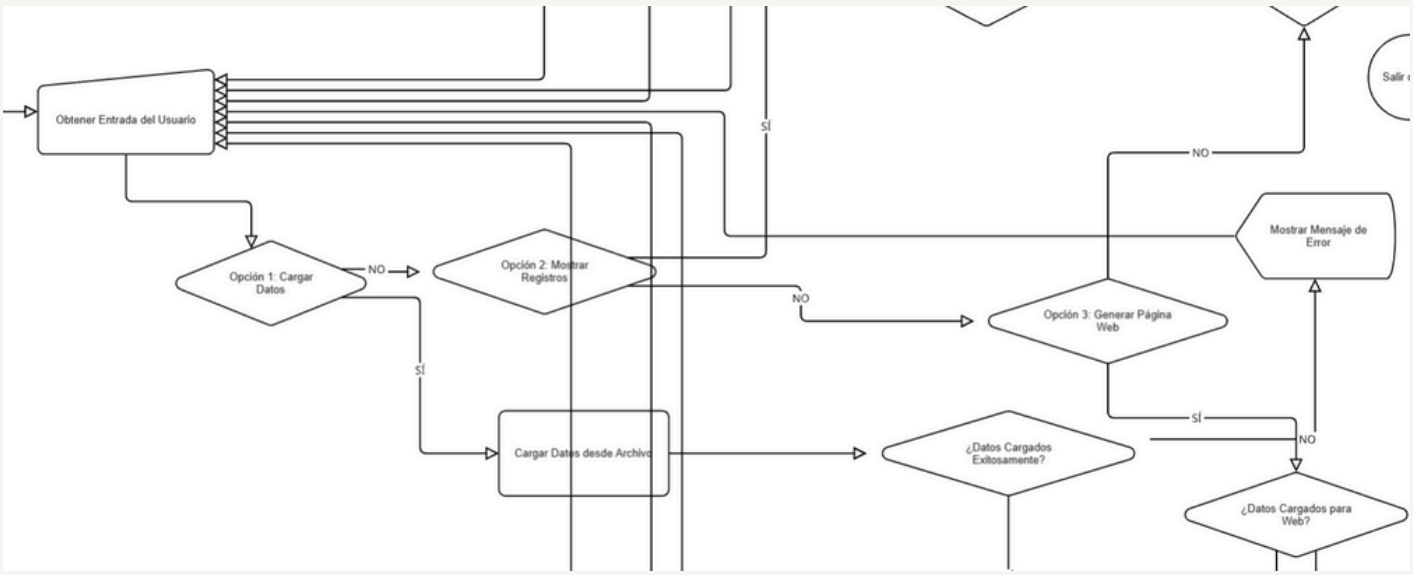
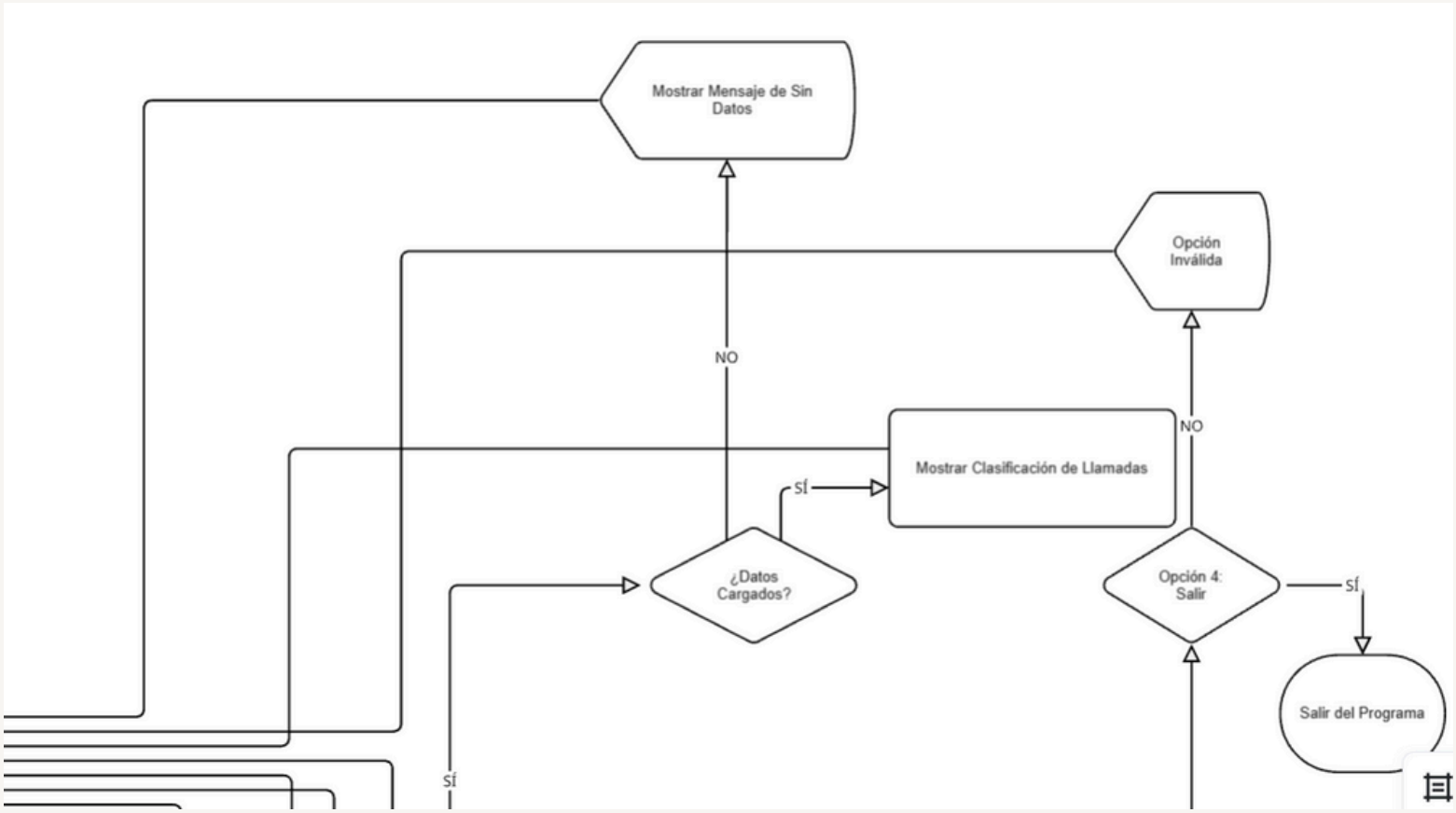
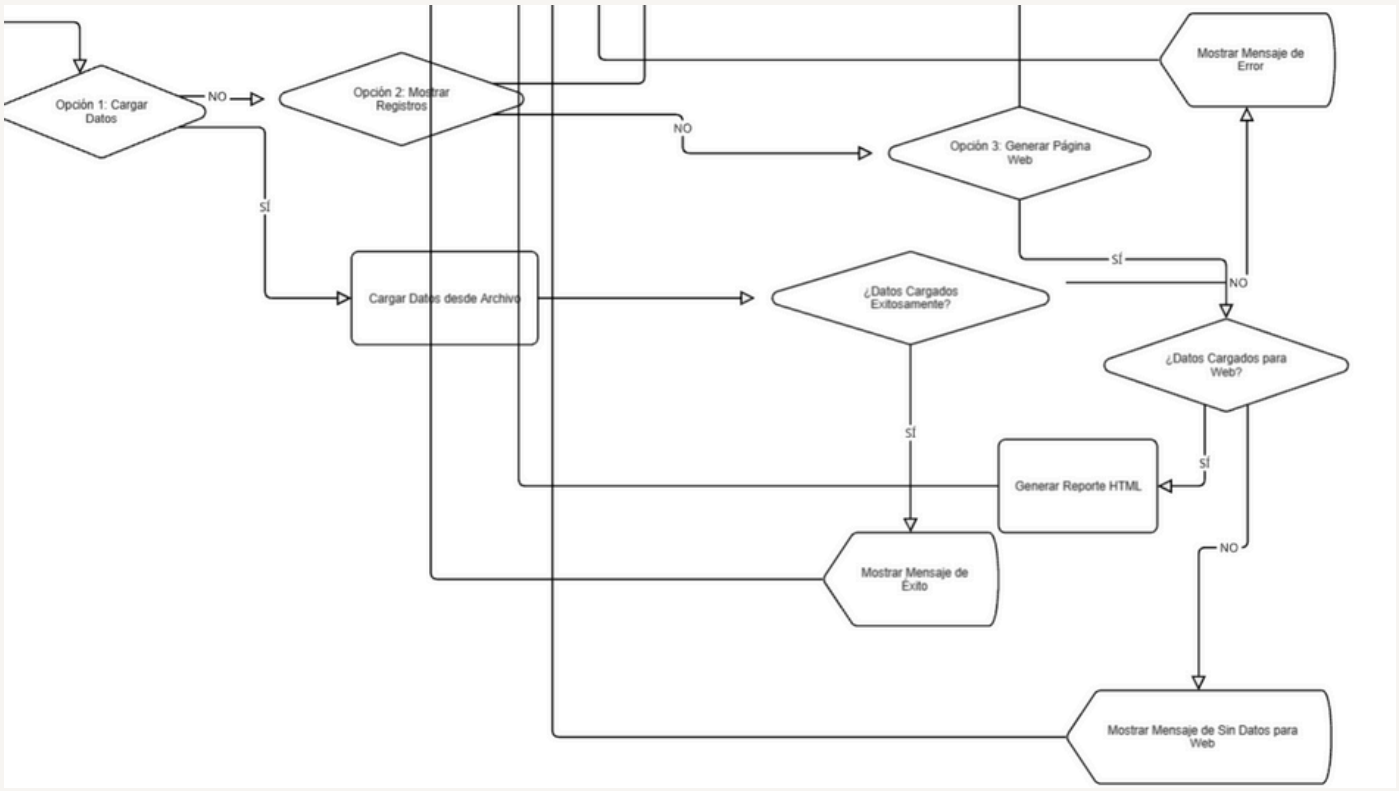


Diagrama de flujo






Mostrar Mensaje de Sin Datos
Opción Inválida
NO
NO
Mostrar Clasificación de Llamadas
SÍ
¿Datos Cargados?
Opción 4: Salir
SÍ
Salir del Programa
SÍ


Mostrar Mensaje de Error
Opción 1: Cargar Datos
NO
Opción 2: Mostrar Registros
NO
Opción 3: Generar Página Web
SÍ
SÍ
Cargar Datos desde Archivo
¿Datos Cargados Exitosamente?
NO
¿Datos Cargados para Web?
SÍ
SÍ
Generar Reporte HTML
Mostrar Mensaje de Éxito
NO
Mostrar Mensaje de Sin Datos para Web