MANUAL TÉCNICO

Simulador de CallCenter JavaScript

LFP

CARNET 202407596

NOMBRE: Jefferson Ajcúc

I. INTRODUCCION

La práctica “USAC Taller Automotriz” se desarrolla en Java siguiendo el patrón Modelo-Vista-Controlador (MVC). El sistema permite:

- Gestionar un inventario de repuestos y servicios (CRUD y carga masiva desde archivos .tmr y .tms).
- Registrar clientes y sus automóviles (con carga masiva desde .tmca), mostrando en todo momento el progreso de cada orden de trabajo mediante hilos que simulan tiempos de espera, servicio y entrega.
- Persistir datos entre ejecuciones a través de serialización en archivos binarios, y generar facturas y reportes PDF con gráficos de pastel y barras para analizar clientes, repuestos y servicios.
- Utilizar Swing para la interfaz gráfica, iText (o similar) para la creación de PDFs, y librerías de gráficos (p. ej. JFreeChart) para visualizar estadísticas.

Este manual describe la arquitectura general, las decisiones de diseño, las funcionalidades implementadas y las tecnologías empleadas, facilitando su comprensión, mantenimiento y futura ampliación.

II. OBJETIVOS

General

- Desarrollar un programa que procese entradas de texto para simular registros de llamadas en un CallCenter utilizando Programación Orientada a Objetos y estructuras de datos.

Específicos

- Comprender el manejo de archivos en JavaScript.
- Implementar estructuras de datos para manejar información.
- Desarrollar habilidades y conocimientos en lógica de programación.
- Implementar reportes gráficos para visualización de resultados.

III. Dirigido

Este manual está orientado a desarrolladores de software, ingenieros de sistemas, y profesionales de TI interesados en entender a fondo el diseño y la lógica detrás del sistema. Está diseñado para aquellos que buscan mantener, ampliar o adaptar el sistema, así como para aquellos interesados en aprender mas sobre la manipulación de cadenas, manejo de archivos CSV y generación de reportes en JavaScript de consola.

IV. Especificación Técnica

Requisitos de Hardware:

- Computadora de escritorio o portátil.
- Al menos 4 GB de memoria RAM.
- Disco duro con al menos 250 GB de capacidad.
- Procesador Intel Core i3 o superior.
- Resolución gráfica mínima de 1024 x 768 píxeles.

Requisitos de Software:

1. Sistema Operativo

- Windows 10 o superior.
- Linux (cualquier distribución moderna: Ubuntu, Fedora, Debian).
- macOS 11 o superior.
- (en otras palabras, mientras no sea Windows XP o una tostadora con Wi-Fi, todo irá bien).

2. Intérprete de JavaScript

- Node.js versión 18.x o superior.
- Incluye npm (Node Package Manager), que se instala automáticamente con Node.js.
- (no vale tener Node versión “prehistórica”, porque si el programa pide un async/await, tu PC podría mirarte raro).

3. Editor de Código Recomendado

- Visual Studio Code (última versión disponible).
- Extensiones opcionales:
  - ESLint (para mantener el código tan ordenado como un archivador de biblioteca).
  - Prettier (porque hasta el código merece verse bonito).

4. Navegador Web

- Cualquiera de los principales (Chrome, Firefox, Edge, Safari) para abrir los reportes HTML.
- (si usas Internet Explorer, el reporte podría tardar más que una llamada de atención al banco en lunes por la mañana).

5. Terminal / Consola

- CMD o PowerShell en Windows.
- Bash/Zsh en Linux o macOS.
- Soporte UTF-8 para que los acentos y caracteres especiales no aparezcan como jeroglíficos.

V. Descripción de los métodos creados

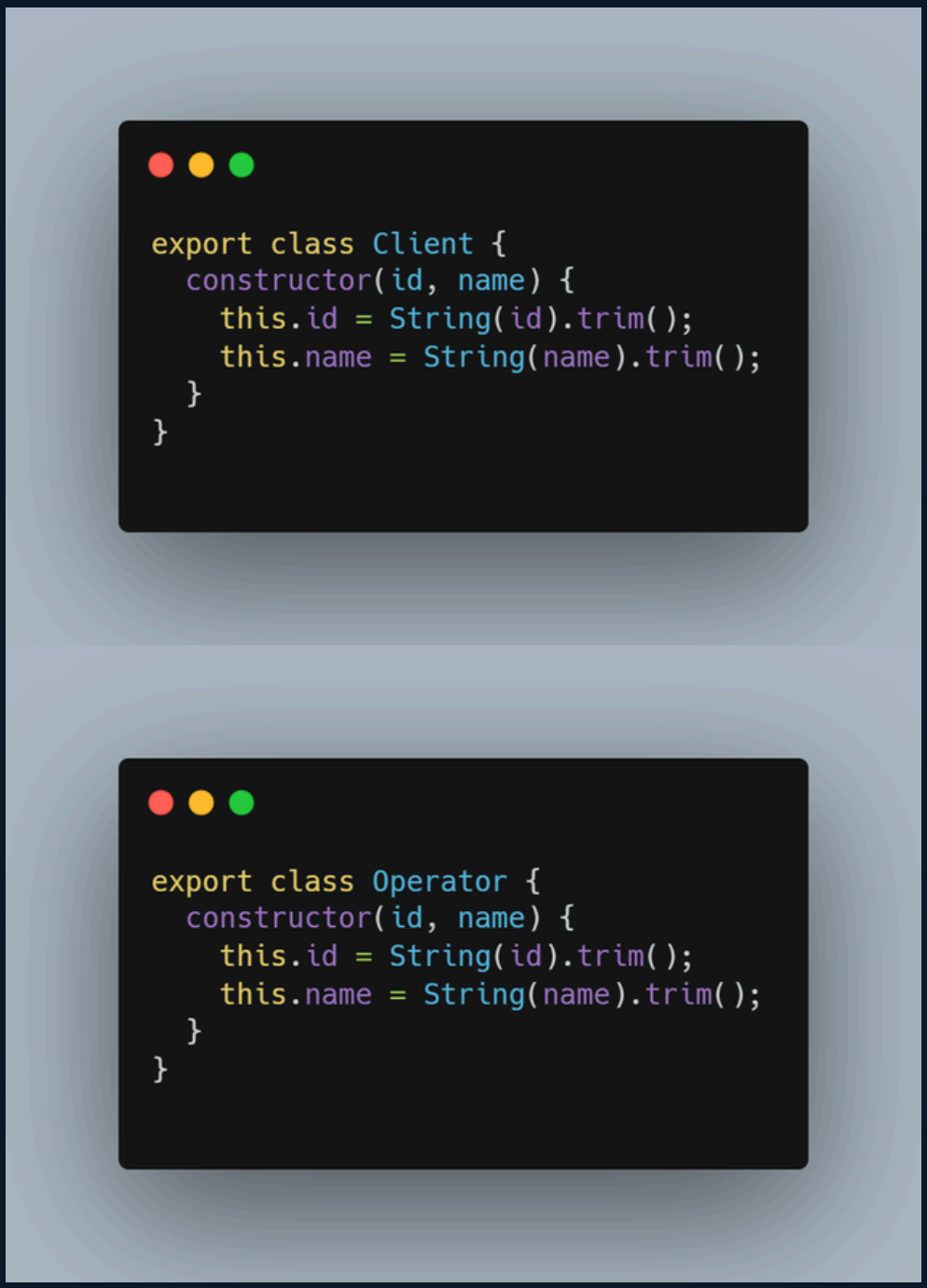
Clase / Módulo	Método / Función	Firma	Descripción
utils/csv	parseCSV	parseCSV(text: string): { headers:	Analiza manualmente el texto de un archivo CSV
	detectDelimiter	detectDelimiter(line: string): string	Detecta automáticamente el
utils/filesystem	ensureDir	ensureDir(dirPath: string): Promise<void>	Verifica y crea un directorio de forma
	writeFileSafe	writeFileSafe(path: string, content: string):	Escribe un archivo de texto creando
	readTextFile	readTextFile(path: string): Promise<string>	Lee un archivo de texto completo en memoria.
utils/formatting	formatPercent	formatPercent(num: number): string	Convierte un número decimal en porcentaje
	nowIsoLocal	nowIsoLocal(): string	Genera una marca de tiempo legible en
models/Call	classification	get classification(): string	Devuelve la clasificación de una llamada (Buena,
store/dataStore	upsertOperator	upsertOperator(id: string, name: string):	Inserta un operador nuevo o actualiza su
	upsertClient	upsertClient(id: string, name: string): void	Inserta un cliente nuevo o actualiza su nombre si
	addCall	addCall(operatorId: string, clientId: string,	Agrega una llamada al registro en memoria.
	getOperatorsArray	getOperatorsArray(): Operator[]	Devuelve un arreglo ordenado de
services/statistics	histogramByStars	histogramByStars(): Record<number,	Devuelve un histograma de llamadas
	percentByClassification	percentByClassification(): { Buena: number,	Calcula el porcentaje de llamadas buenas,
services/performance	operatorPerformance	operatorPerformance(): { id: string, name: string,	Calcula el porcentaje de atención de cada
io/csvLoader	loadCsv	loadCsv(filePath: string): Promise<void>	Carga un archivo CSV, interpreta los datos y los guarda en memoria.
io/htmlExporter	exportHistorial	exportHistorial(path: string): Promise<void>	Exporta en HTML el historial de llamadas.
	exportOperadores	exportOperadores(path: string): Promise<void>	Exporta en HTML el listado de operadores.
	exportClientes	exportClientes(path: string): Promise<void>	Exporta en HTML el listado de clientes.
	exportRendimiento	exportRendimiento(path: string): Promise<void>	Exporta en HTML el rendimiento de los operadores.
index (menú)	question	question(rl: Interface, prompt: string): Promise<string>	Func leer entradas del usuario en la consola.

```
function splitCSVLine(line, delimiter) {
  const parts = [];
  let buf = '';
  let inQuotes = false;

  for (let i = 0; i < line.length; i++) {
    const ch = line[i];
    if (ch === '"') {
      // handle escaped quotes "" -> "
      if (inQuotes && line[i + 1] === '"') {
        buf += '"'; i++;
      } else {
        inQuotes = !inQuotes;
      }
    } else if (ch === delimiter && !inQuotes) {
      parts.push(buf); buf = '';
    } else {
      buf += ch;
    }
  }
  parts.push(buf);
  return parts;
}
```

```
export class Call {
  constructor(operatorId, clientId, stars) {
    this.operatorId = String(operatorId).trim();
    this.clientId = String(clientId).trim();
    this.stars = Number(stars) || 0;
    if (this.stars < 0) this.stars = 0;
    if (this.stars > 5) this.stars = 5;
  }

  get classification() {
    if (this.stars >= 4) return 'Buena';
    if (this.stars >= 2) return 'Media';
    return 'Mala';
  }
}
```


export class Client {
constructor(id, name) {
this.id = String(id).trim();
this.name = String(name).trim();
}
}
export class Operator {
constructor(id, name) {
this.id = String(id).trim();
this.name = String(name).trim();
}
}

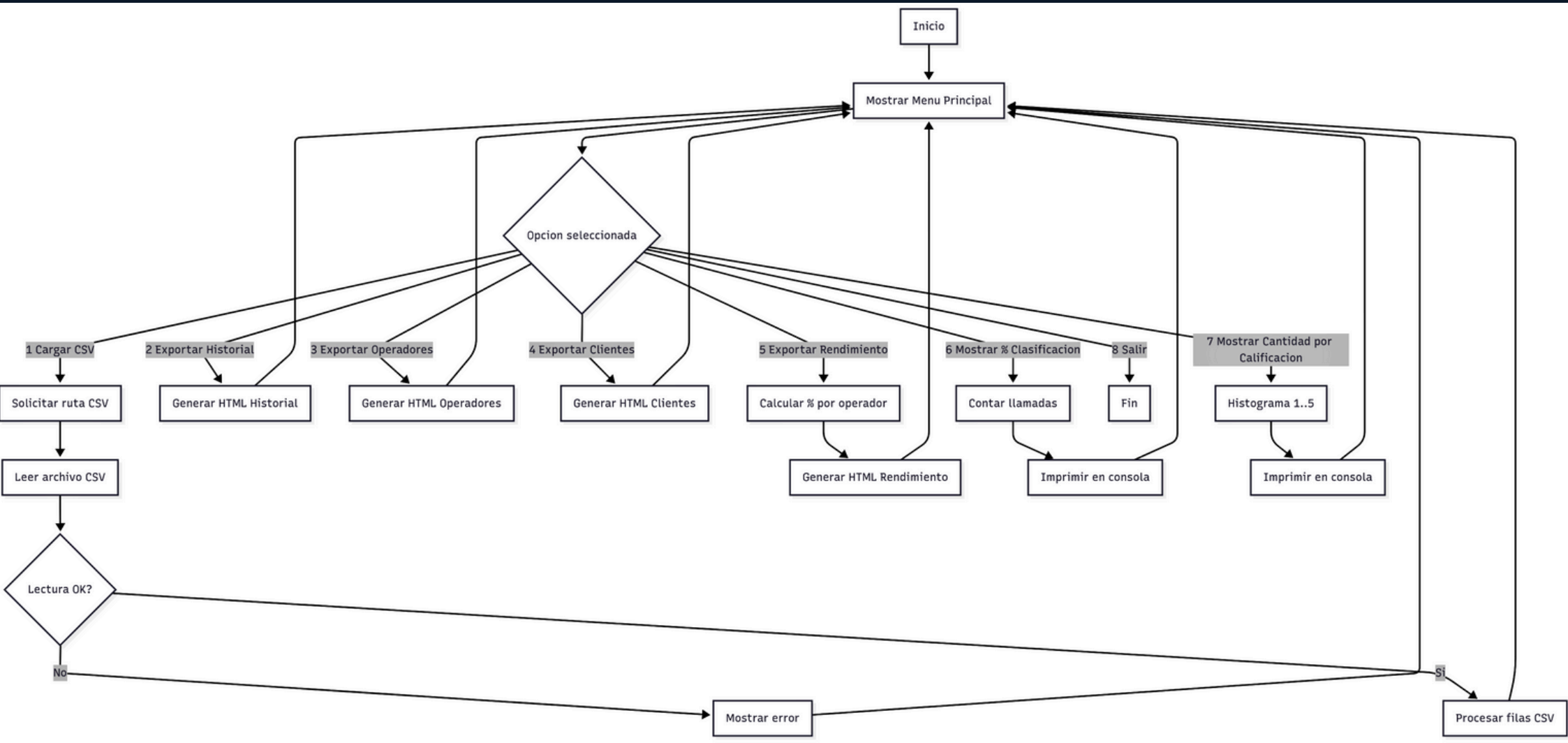
```
export function operatorPerformance() {
  const totals = new Map(); // operatorId -> count
  for (const c of store.calls) {
    totals.set(c.operatorId, (totals.get(c.operatorId) || 0) + 1);
  }
  const totalCalls = store.getTotalCalls() || 1;
  const result = [];
  for (const op of store.getOperatorsArray()) {
    const count = totals.get(op.id) || 0;
    result.push({
      id: op.id,
      name: op.name,
      calls: count,
      attentionPercent: (count * 100) / totalCalls
    });
  }
  return result.sort((a,b) => Number(b.attentionPercent) - Number(a.attentionPercent));
}
```

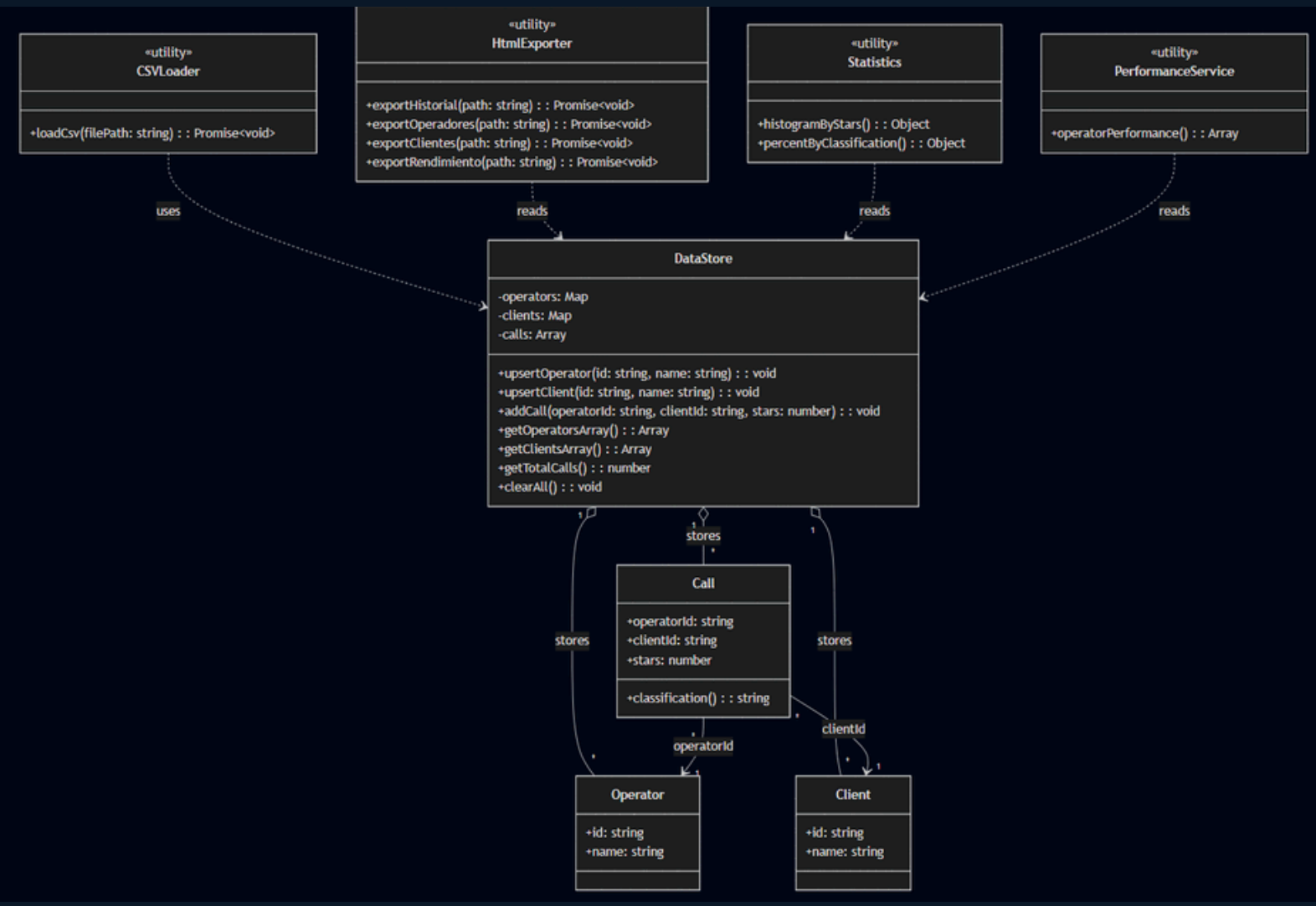
```
export function histogramByStars() {
  const h = {1:0,2:0,3:0,4:0,5:0};
  for (const c of store.calls) {
    if (c.stars >= 1 && c.stars <= 5) h[c.stars]++;
    else if (c.stars === 0) { /* ignore 0-star into histogram 1..5 */ }
  }
  return h;
}
```

```
export function percentByClassification() {
  let good=0, med=0, bad=0;
  for (const c of store.calls) {
    if (c.stars >= 4) good++;
    else if (c.stars >= 2) med++;
    else bad++;
  }
  const total = store.getTotalCalls() ||
1;return {
    Buena: (good*100)/total,
    Media: (med*100)/total,
    Mala:  (bad*100)/total
  };
}
```

III. Diccionario de métodos usados (principales)

Método / API	Parámetros principales	Retorno	Función resumida
fs/promises.readFile	(path, 'utf-8')	Promise<string>	Lee el contenido del archivo CSV.
fs/promises.writeFile	(path, data, 'utf-8')	Promise<void>	Escribe los reportes HTML en disco.
path.join / path.resolve	(...segments)	string	Construye rutas absolutas o relativas de archivos.
readline.createInterface	({ input, output })	Interface	Crea la interfaz de consola para el menú.
Interface.question	(prompt, cb)	void	Pide datos al usuario en consola.
Map.set / Map.get	(key, value) / (key)	value	Almacena y consulta operadores y clientes en memoria.
Array.prototype.sort	(compareFn)	this	Ordena operadores y clientes para los reportes.
Array.prototype.reduce	(fn, init)	any	Cuenta las estrellas (x) en las llamadas.
String.prototype.split	(sep)	string[]	Divide líneas y campos del CSV.
String.prototype.trim	()	string	Limpia espacios en los valores leídos del archivo.


Inicio
Mostrar Menu Principal
Opcion seleccionada
1 Cargar CSV
2 Exportar Historial
3 Exportar Operadores
4 Exportar Clientes
5 Exportar Rendimiento
6 Mostrar % Clasificacion
8 Salir
7 Mostrar Cantidad por Calificacion
Solicitar ruta CSV
Generar HTML Historial
Generar HTML Operadores
Generar HTML Clientes
Calcular % por operador
Contar llamadas
Fin
Histograma 1..5
Leer archivo CSV
Generar HTML Rendimiento
Imprimir en consola
Imprimir en consola
Lectura OK?
No
Si
Mostrar error
Procesar filas CSV


«utility»
CSVLoader
+loadCsv(filePath: string) : : Promise<void>
«utility»
HtmlExporter
+exportHistorial(path: string) : : Promise<void>
+exportOperadores(path: string) : : Promise<void>
+exportClientes(path: string) : : Promise<void>
+exportRendimiento(path: string) : : Promise<void>
«utility»
Statistics
+histogramByStars() : : Object
+percentByClassification() : : Object
«utility»
PerformanceService
+operatorPerformance() : : Array
uses
reads
reads
reads
DataStore
-operators: Map
-clients: Map
-calls: Array
+upsertOperator(id: string, name: string) : : void
+upsertClient(id: string, name: string) : : void
+addCall(operatorId: string, clientId: string, stars: number) : : void
+getOperatorsArray() : : Array
+getClientsArray() : : Array
+getTotalCalls() : : number
+clearAll() : : void
1
1
1
stores
*
stores
stores
Call
+operatorId: string
+clientId: string
+stars: number
+classification() : : string
*
*
operatorId
clientId
*
*
1
1
Operator
+id: string
+name: string
Client
+id: string
+name: string