



Informe de Análisis y Diseño – Evaluación Práctica Progreso 1

Juan Aristizábal

Integración de Sistemas

Facultad de Ingeniería y Ciencias Aplicadas, Universidad De Las Américas

Abril 23, 2025

Informe de Análisis y Diseño – Evaluación Práctica Progreso 1

1. Identificación del Problema

Actualmente, BioNet enfrenta serios desafíos de integración entre sus distintos laboratorios clínicos distribuidos geográficamente. Los principales riesgos del sistema actual incluyen el manejo manual de archivos CSV, la falta de validación automática, la sobrescritura de datos y los conflictos de concurrencia en la base de datos central. Cada laboratorio genera archivos .csv diariamente, los cuales son transferidos manualmente a un servidor FTP compartido. Este proceso no solo es propenso a errores humanos, sino que también carece de trazabilidad y control. Además, no hay mecanismos que validen si los archivos están completos o correctamente formateados antes de su integración. Esto ha resultado en múltiples incidentes de datos duplicados, sobrescritos o inconsistencias cuando varios procesos acceden simultáneamente a la base de datos. En conjunto, estos problemas afectan la integridad y confiabilidad del sistema central de gestión de resultados.

2. Justificación del Uso de Patrones

- **Transferencia de Archivos**

Se implementó un sistema automático que vigila continuamente una carpeta local (input-labs/) para detectar nuevos archivos .csv. Este patrón reemplaza la transferencia por FTP y elimina la intervención manual, reduciendo errores y mejorando la trazabilidad del proceso.

- **Base de Datos Compartida**

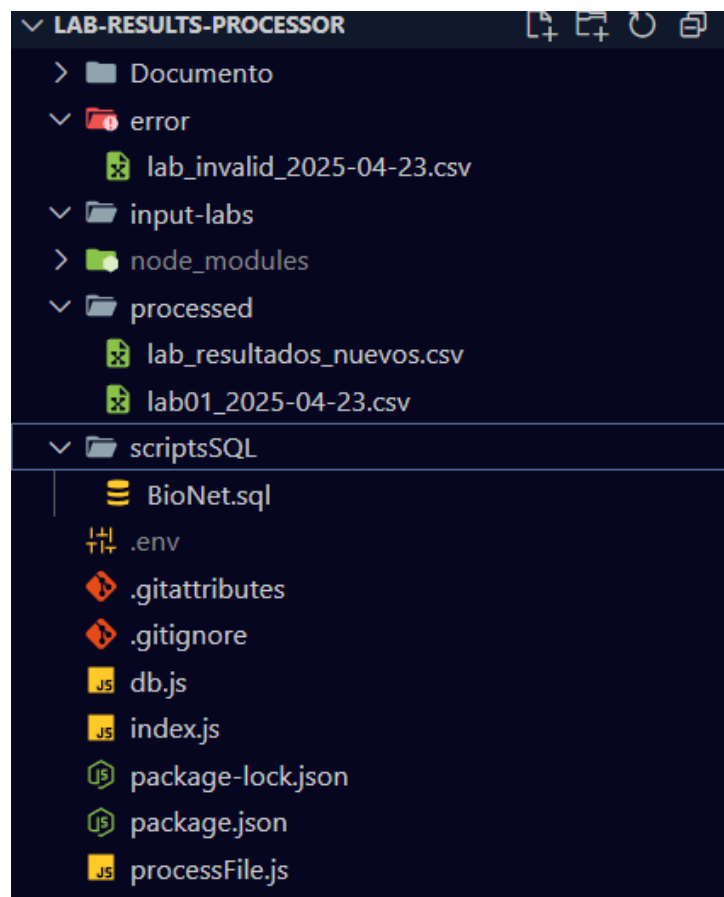
Para consolidar todos los resultados en un único punto de verdad, se utilizó una base de datos SQL Server (bionet_db). Esta base está protegida mediante restricciones de unicidad y un trigger de auditoría. El patrón de base de datos compartida facilita la sincronización entre laboratorios y el sistema central, asegurando integridad y consistencia.

3. Diseño de Alto Nivel de la Solución

La solución propuesta para la integración de resultados clínicos en la empresa BioNet está diseñada para funcionar de manera automática, modular y resiliente, con el objetivo de consolidar información proveniente de distintos laboratorios hacia un sistema centralizado. El enfoque general evita depender de procesos manuales y reduce al mínimo los riesgos de duplicación, errores de formato o sobrescritura de datos. El flujo inicia cuando un

laboratorio genera un archivo .csv con los resultados de los exámenes realizados en una jornada. Este archivo es colocado en una carpeta local compartida llamada input-labs/, la cual es monitoreada de forma constante por una aplicación automatizada. En cuanto se detecta un nuevo archivo en esta carpeta, el sistema lo valida, asegurándose de que su estructura sea correcta y contenga datos completos. Una vez validado, el archivo es procesado línea por línea. Para cada resultado, el sistema verifica si ya existe en la base de datos central, comparando el paciente_id, el tipo_examen y la fecha_examen. Si el registro no existe, se inserta en la tabla resultados_examenes. Si ya existe, se omite para evitar duplicados. Además, cualquier inserción o actualización genera automáticamente un registro en la tabla log_cambios_resultados gracias a un trigger de auditoría. Después del procesamiento, el archivo es movido a una carpeta de clasificación. Si fue procesado con éxito, se traslada a processed/; si presentó errores en su formato o contenido, se mueve a error/ para su revisión posterior. Este mecanismo garantiza orden, trazabilidad y control de calidad.

4. Estructura de Carpetas



- **error/:** Archivos .csv que fueron rechazados por estar mal formateados o incompletos.
- **input-labs/:** Carpeta vigilada por el sistema donde se colocan los archivos .csv nuevos a procesar.
- **processed/:** Archivos .csv que ya fueron validados e insertados correctamente en la base de datos.
- **scriptsSQL/:** Contiene los scripts .sql necesarios para crear la base de datos, tablas y triggers.
- **.env:** Variables de entorno con las credenciales de conexión a SQL Server (no debe subirse a Git).
- **db.js:** Configura y exporta la conexión al servidor de base de datos SQL Server usando mssql.
- **index.js:** Script principal que monitorea la carpeta input-labs/ y lanza el procesamiento de archivos.
- **processFile.js:** Lógica que valida, evita duplicados y guarda los resultados del .csv en la base de datos.
- **BioNet.sql:** Script con todo lo necesario para crear la base de datos bionet_db, sus tablas y triggers.

5. Flujo de Integración

- **Generación del archivo CSV:** Cada laboratorio crea diariamente un archivo .csv con los resultados de los exámenes realizados. Este archivo contiene columnas como: laboratorio_id, paciente_id, tipo_examen, resultado y fecha_examen.
- **Colocación en carpeta input-labs/:** El archivo se coloca en la carpeta local input-labs/. Este directorio es monitoreado automáticamente por el sistema usando fs.watch.
- **Detección automática del archivo:** El script index.js detecta nuevos archivos .csv cuando se crean o copian en input-labs/.
- **Validación de estructura:** El archivo es leído por processFile.js y validado:
 - Debe tener más de una línea (cabecera + datos)
 - Cada fila debe contener todos los campos requeridos

- **Verificación de duplicados:** Por cada fila, se consulta la base de datos para verificar si ya existe un resultado con la misma combinación de paciente_id, tipo_examen y fecha_examen.
- **Inserción en la base de datos:** Si el resultado no existe, se inserta en la tabla resultados_exámenes. Si ya existe, se omite y se muestra en consola como “Duplicado ignorado”.
- **Registro de auditoría con trigger:** Cada inserción o actualización activa el trigger trg_log_cambios, que registra automáticamente la operación en log_cambios_resultados.
- **Clasificación del archivo:**
 - Si fue procesado correctamente, se mueve a la carpeta processed/.
 - Si tiene errores (estructura incorrecta, filas vacías, etc.), se mueve a la carpeta error/.

6. Esquema de Base de Datos Sugerido

log cambios resultados	
id	
operacion	
paciente_id	
tipo_examen	
fecha	

resultados examenes	
id	
laboratorio_id	
paciente_id	
tipo_examen	
resultado	
fecha_examen	

```

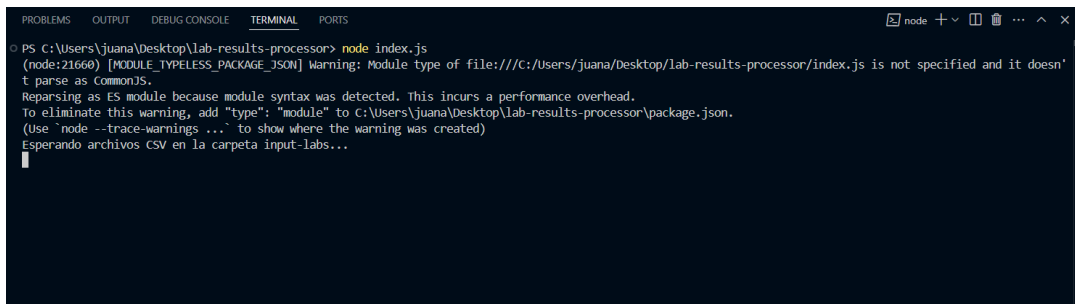
CREATE TABLE resultados_examenes (
    id INT IDENTITY(1,1) PRIMARY KEY,
    laboratorio_id VARCHAR(50),
    paciente_id VARCHAR(50),
    tipo_examen VARCHAR(100),
    resultado TEXT,
    fecha_examen DATE,
    CONSTRAINT UQ_Examen UNIQUE (paciente_id, tipo_examen, fecha_examen)
);

CREATE TABLE log_cambios_resultados (
    id INT IDENTITY(1,1) PRIMARY KEY,
    operacion VARCHAR(10),
    paciente_id VARCHAR(50),
    tipo_examen VARCHAR(100),
    fecha DATETIME DEFAULT GETDATE()
);

```

7. Capturas de ejecución funcional

a. Esperando Archivos

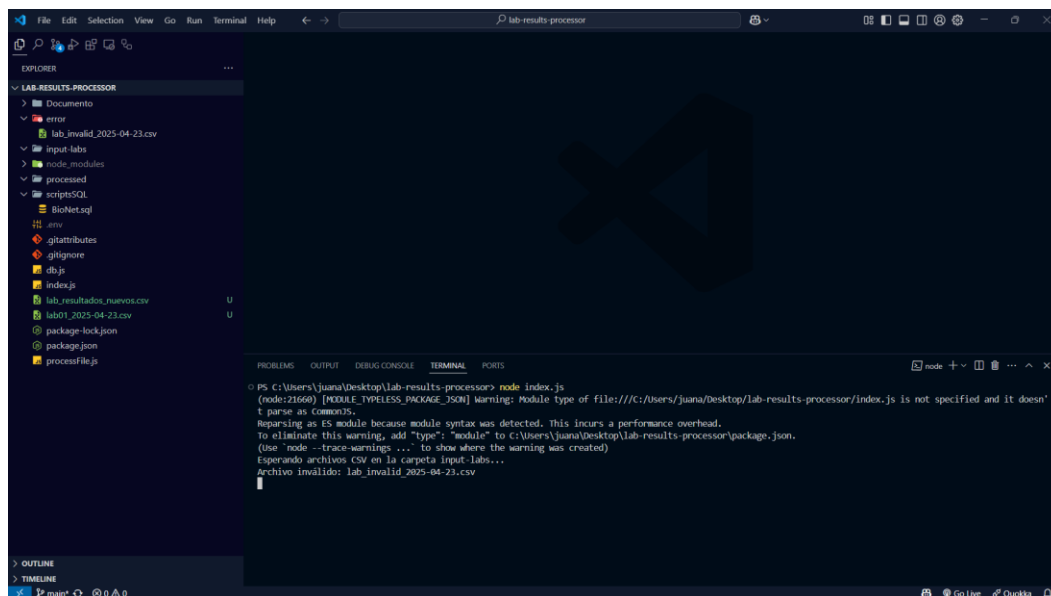


```

PS C:\Users\juana\Desktop\lab-results-processor> node index.js
(node:21660) [MODULE_TYPELESS_PACKAGE_JSON] Warning: Module type of file:///C:/Users/juana/Desktop/lab-results-processor/index.js is not specified and it doesn't
t parse as CommonJS.
Reparsing as ES module because module syntax was detected. This incurs a performance overhead.
To eliminate this warning, add "type": "module" to C:\Users\juana\Desktop\lab-results-processor\package.json.
(Use 'node --trace-warnings ...' to show where the warning was created)
Esperando archivos CSV en la carpeta input-labs...

```

b. Archivo Incorrecto



```

PS C:\Users\juana\Desktop\lab-results-processor> node index.js
(node:21660) [MODULE_TYPELESS_PACKAGE_JSON] Warning: Module type of file:///C:/Users/juana/Desktop/lab-results-processor/index.js is not specified and it doesn't
t parse as CommonJS.
Reparsing as ES module because module syntax was detected. This incurs a performance overhead.
To eliminate this warning, add "type": "module" to C:\Users\juana\Desktop\lab-results-processor\package.json.
(Use 'node --trace-warnings ...' to show where the warning was created)
Esperando archivos CSV en la carpeta input-labs...
Archivo invalido: lab_invalid_2025-04-23.csv

```

c. Archivos Correctos (Los duplicados son ignorados)

The screenshot shows a VS Code editor with a file named `lab_resultados_nuevos.csv` open. The file contains a CSV with columns: `laboratorio_id`, `paciente_id`, `tipo_examen`, `resultado`, and `fecha_examen`. The data includes various lab tests like GLUCOSA, COLESTEROL, TRIGLICERIDOS, UREA, CREATININA, HEMOGLOBINA, CALCIO, POTASIO, SODIO, FOSFATASA, and HEMOGLOBINA. The terminal window shows the execution of a script that processes these files, inserting data into a database. The logs indicate that some records were already present and were ignored (e.g., "Insertado: PAC201 - GLUCOSA", "Procesando archivo: lab01_2025-04-23.csv", "Insertado: PAC202 - COLESTEROL", "Insertado: PAC001 - GLUCOSA", "Insertado: PAC203 - TRIGLICERIDOS", "Insertado: PAC002 - GLUCOSA", "Insertado: PAC204 - UREA", "Insertado: PAC004 - GLUCOSA", "Insertado: PAC205 - CREATININA", "Insertado: PAC005 - GLUCOSA", "Insertado: PAC206 - HEMOGLOBINA", "Insertado: PAC001 - COLESTEROL", "Insertado: PAC207 - CALCIO", "Insertado: PAC002 - HEMOGLOBINA", "Duplicado Ignorado: PAC003 - GLUCOSA", "Insertado: PAC208 - POTASIO").

d. Ya se muestran los datos en la base de datos

The screenshot shows the SQL Server Enterprise Manager interface. A query is executed in the "Query Editor" window, displaying the results of a query that selects data from the `log_cambios_resultados` table. The query is:

```
SELECT * FROM log_cambios_resultados
```

. The results are displayed in a table with columns: `id`, `laboratorio_id`, `paciente_id`, `tipo_examen`, `resultado`, and `fecha_examen`. The data includes various lab tests like GLUCOSA, COLESTEROL, TRIGLICERIDOS, UREA, CREATININA, HEMOGLOBINA, CALCIO, POTASIO, SODIO, FOSFATASA, and HEMOGLOBINA. The status bar at the bottom indicates "Query executed successfully."

Capturas del Código

db.js: Este archivo se encarga de configurar la conexión a la base de datos SQL Server utilizando las credenciales definidas en el archivo .env. Crea una instancia reusable (poolPromise) de ConnectionPool de la librería mssql, permitiendo que otros módulos del proyecto ejecuten consultas SQL sin tener que reconectarse cada vez. Además, exporta tanto el objeto sql como la conexión activa para que puedan ser utilizados en otros scripts del sistema.

```
1  import sql from "mssql";
2  import dotenv from "dotenv";
3
4  dotenv.config();
5
6  const dbConfig = {
7    user: process.env.DB_USER,
8    password: process.env.DB_PASSWORD,
9    server: process.env.DB_SERVER,
10   database: process.env.DB_DATABASE,
11   options: {
12     encrypt: false,
13     trustServerCertificate: true,
14   },
15 };
16
17 export const poolPromise = new sql.ConnectionPool(dbConfig).connect();
18 export { sql };
19
```

index.js: Este es el punto de entrada del sistema. Su función principal es monitorear la carpeta input-labs/ en busca de nuevos archivos .csv. Utiliza fs.watch para detectar cambios, y cuando se añade un nuevo archivo, verifica su existencia y lo pasa a la función de procesamiento. Este script mantiene al sistema en ejecución continua, esperando archivos nuevos para iniciar automáticamente su tratamiento.


```

1  import fs from "fs";
2  import path from "path";
3  import processCsvFile from "../processFile.js";
4
5  const watchFolder = "input-labs";
6
7  fs.watch(watchFolder, async (eventType, filename) => {
8      if (filename.endsWith(".csv") && eventType === "rename") {
9          const fullPath = path.join(watchFolder, filename);
10         if (fs.existsSync(fullPath)) {
11             await processCsvFile(fullPath);
12         }
13     }
14 });
15
16 console.log("Esperando archivos CSV en la carpeta input-labs...");
17

```

processFile.js: Este módulo contiene la lógica de procesamiento de los archivos .csv. Primero valida que el archivo tenga al menos una línea de datos, luego lo lee fila por fila usando csv-parser. Para cada fila, verifica si ya existe un registro en la base de datos con la misma combinación de paciente_id, tipo_examen y fecha_examen. Si el registro es nuevo, lo inserta en la tabla resultados_examenes; si es duplicado, lo omite. Finalmente, mueve el archivo a processed/ si fue exitosamente tratado, o a error/ si tuvo problemas de formato.

```

6  const isValidCsv = async (filePath) => {
7    const lines = await fs.readFile(filePath, "utf8");
8    return lines.trim().split("\n").length > 1;
9  };
10
11  const processCsvFile = async (filePath) => {
12    if (!(await isValidCsv(filePath))) {
13      console.log(`Archivo inválido: ${path.basename(filePath)}`);
14      await fs.move(filePath, path.join("error", path.basename(filePath)), {
15        overwrite: true,
16      });
17      return;
18    }
19
20    console.log(`Procesando archivo: ${path.basename(filePath)}`);
21
22    const rows = [];
23    const pool = await poolPromise;
24
25    fs.createReadStream(filePath)
26      .pipe(csv())
27      .on("data", (row) => rows.push(row))
28      .on("end", async () => {
29        for (const row of rows) {
30          const { paciente_id, tipo_examen, fecha_examen } = row;
31
32          const checkQuery = `
33            SELECT 1 FROM resultados_examenes
34            WHERE paciente_id = @paciente_id AND tipo_examen = @tipo_examen AND fecha_examen = @fecha_examen
35          `;
36
37          const result = await pool
38            .request()
39            .input("paciente_id", sql.VarChar, paciente_id)
40            .input("tipo_examen", sql.VarChar, tipo_examen)
41            .input("fecha_examen", sql.Date, fecha_examen)
42            .query(checkQuery);
43
44          if (result.recordset.length === 0) {
45            const insertQuery = `
46              INSERT INTO resultados_examenes (laboratorio_id, paciente_id, tipo_examen, resultado, fecha_examen)
47              VALUES (@laboratorio_id, @paciente_id, @tipo_examen, @resultado, @fecha_examen)
48            `;

```

```

50      .request()
51      .input("laboratorio_id", sql.VarChar, row.laboratorio_id)
52      .input("paciente_id", sql.VarChar, paciente_id)
53      .input("tipo_examen", sql.VarChar, tipo_examen)
54      .input("resultado", sql.VarChar, row.resultado)
55      .input("fecha_examen", sql.Date, fecha_examen)
56      .query(insertQuery);
57
58      console.log(`Insertado: ${paciente_id} - ${tipo_examen}`);
59    } else {
60      console.log(`Duplicado ignorado: ${paciente_id} - ${tipo_examen}`);
61    }
62  }
63
64  await fs.move(filePath, path.join("processed", path.basename(filePath)), {
65    overwrite: true,
66  });
67  });
68  };
69
70  export default processCsvFile;
71

```