

Materia: Programación Paralela.

Título: Sistema para el análisis en paralelo de logs.

Integrantes y Matriculas:

Luis David Marte Vásquez 2023-1165

Luis Ángel Sánchez 2023-1681

Jonathan José frías Martínez 2023-1117

Líder: Gilberto Yunion Hernández De Los Santos 2023-1211

Carrera: Desarrollo de software

Nombre del docente: Erick Leonardo Pérez Veloz.

Fecha: 18/8/2025

Índice

1. Introducción	3
2. Descripción del Problema	4
3. Cumplimiento de los Requisitos del Proyecto	5
4. Diseño de la Solución	6
5. Implementación Técnica	10
6. Evaluación de Desempeño	14
7. Trabajo en Equipo	16
8. Conclusiones	17
9. Referencias	18
10. Anexos	19
GitHub: https://github.com/GilbertoHernandez150/Proyecto-Final-Paralela	23

1. Introducción

Presentación general del proyecto

El proyecto desarrollado consiste en un sistema para el **análisis paralelo de logs del sistema a través de conexión remota**. La aplicación permite conectarse a equipos con sistemas operativos Windows o Linux utilizando protocolos de **SSH y SFTP**, ejecutar comandos o scripts que generan archivos de logs del sistema, transferir dichos archivos al servidor local y analizarlos de forma **secuencial y paralela**.

El análisis se centra en identificar eventos relevantes como **errores, advertencias e información general** y clasificarlos. Entre sus características principales se encuentran la **ejecución simultánea de múltiples tareas** para procesar los logs en paralelo, el uso de **mecanismos de sincronización** para combinar resultados parciales de cada hilo de ejecución, y la **comparación de rendimiento** entre el análisis secuencial y paralelo mediante métricas como el **speedup** y la **eficiencia**.

Asimismo, el sistema se acompaña de una **interfaz web** que permite visualizar gráficamente los resultados a través de un dashboard dinámico, facilitando la interpretación de los datos y la toma de decisiones en escenarios de monitoreo, mantenimiento y detección temprana de fallos.

Justificación del tema elegido

La elección de este tema se fundamenta en los siguientes aspectos:

✚ Relevancia técnica y educativa:

El proyecto aborda conceptos esenciales de la programación concurrente y paralela, aplicando técnicas como **Parallel.ForEach**, sincronización mediante bloqueos y estructuras concurrentes. Esto permite comprender cómo se implementan sistemas robustos y eficientes en escenarios de análisis de datos de gran volumen.

✚ Aplicación en entornos reales:

El análisis de logs es una tarea crítica en entornos laborales donde se requiere supervisar el funcionamiento de **servidores, estaciones de trabajo o aplicaciones empresariales**. El sistema desarrollado simula procesos reales de auditoría y mantenimiento preventivo, siendo aplicable para la detección temprana de fallos y la generación de métricas operativas.

✚ Optimización de procesos y evaluación del rendimiento:

La comparación entre el procesamiento secuencial y paralelo ofrece una plataforma para estudiar el **impacto del paralelismo** en tareas intensivas de análisis de datos. Al medir **speedup** y **eficiencia**, se obtiene una base sólida para comprender la escalabilidad del sistema en escenarios con diferentes cargas de trabajo.

Objetivos (general y específicos)

Objetivo General

Desarrollar un sistema que permita conectarse de forma remota a equipos Windows para **obtener, procesar y analizar archivos de logs** por medio de **SSH**, aplicando técnicas de programación paralela que optimicen el rendimiento y generen métricas interpretables sobre el comportamiento del sistema.

Objetivos Específicos

- ✚ Implementar un mecanismo de conexión remota seguro mediante **SSH y SFTP** para ejecutar comandos y transferir archivos de logs al servidor de análisis.
- ✚ Desarrollar un módulo de **procesamiento secuencial y paralelo** que permita comparar la eficiencia de ambas estrategias sobre archivos de gran tamaño.
- ✚ Aplicar **mecanismos de sincronización** para el manejo seguro de datos compartidos durante la clasificación de eventos en múltiples hilos.
- ✚ Evaluar el rendimiento del sistema mediante **métricas como tiempo de ejecución, speedup y eficiencia**, identificando el impacto del paralelismo bajo diferentes condiciones de carga.
- ✚ Proporcionar un **dashboard interactivo** que muestre gráficamente los resultados del análisis, facilitando la interpretación de métricas y la detección de patrones en los logs.

2. Descripción del Problema

✚ Contexto del problema seleccionado

El problema se centra en el **análisis eficiente de archivos de logs del sistema**, los cuales son generados continuamente en servidores y estaciones de trabajo. Estos logs contienen información crítica sobre eventos del sistema, errores, advertencias y sucesos generales que permiten a los administradores detectar fallos, prevenir incidentes y auditar el correcto funcionamiento de los equipos. El reto radica en que dichos archivos pueden crecer rápidamente en tamaño, dificultando su análisis manual o secuencial, por lo que se requiere una solución que aproveche el **procesamiento paralelo** para obtener resultados en menos tiempo y de manera escalable.

✚ Aplicación del problema en un escenario real

En un entorno real, este problema se refleja en la necesidad de **monitorear servidores y estaciones de trabajo en tiempo real**. Por ejemplo, en un centro de datos o en una empresa, los administradores de TI necesitan identificar de forma rápida cuántos errores críticos se han generado, qué advertencias están

afectando la estabilidad del sistema y qué eventos informativos describen su comportamiento.

Un sistema que se conecte de forma remota a los equipos, recopile sus logs y los procese de forma paralela reduce significativamente los tiempos de análisis y permite **reaccionar a incidentes en menor tiempo**, mejorando la seguridad y la continuidad del servicio.

✚ **Importancia del paralelismo en la solución**

El uso del paralelismo es esencial en este proyecto porque los archivos de logs suelen contener **miles o millones de líneas**, lo que convierte su procesamiento en una tarea intensiva. Mediante la división del archivo en bloques y la ejecución de hilos en paralelo, es posible **clasificar y contar eventos simultáneamente**. Esto no solo acelera el tiempo de análisis, sino que también optimiza el uso de los recursos del sistema, logrando un mejor aprovechamiento de los procesadores disponibles y garantizando resultados más eficientes en comparación con la ejecución secuencial.

3. Cumplimiento de los Requisitos del Proyecto

1. Ejecución simultánea de múltiples tareas

El archivo de log obtenido de un servidor remoto se divide en bloques que son procesados de manera paralela. Cada tarea analiza un subconjunto de líneas para identificar eventos como **errores, advertencias o información general**, mientras otras tareas realizan operaciones de conteo y clasificación, lo que permite un análisis simultáneo del mismo archivo.

2. Necesidad de compartir datos entre tareas

Las tareas deben combinar sus resultados parciales (por ejemplo, el número de errores o advertencias encontrados). Para esto se emplean mecanismos de **sincronización como locks**, garantizando que las estructuras de datos compartidas no sean modificadas por más de un hilo al mismo tiempo.

3. Exploración de diferentes estrategias de paralelización

- **Descomposición de datos:** el archivo de log se divide en secciones independientes que son procesadas en paralelo.
- **Paralelismo de tareas:** además del análisis de bloques de líneas, se realizan operaciones de agregación y clasificación concurrentemente.
- **Sincronización de recursos compartidos:** se emplean estructuras seguras y bloqueos controlados para combinar resultados sin inconsistencias.

- **Control de paralelismo:** se utiliza `MaxDegreeOfParallelism` y `Environment.ProcessorCount` para ajustar el número de hilos a los recursos disponibles del sistema.

4. Escalabilidad con más recursos

El sistema puede aprovechar más **núcleos de CPU** a medida que crece el tamaño de los archivos de log o el número de dispositivos analizados. Esto permite mantener tiempos de análisis aceptables incluso en escenarios de alta carga.

5. Métricas de evaluación del rendimiento

- **Tiempo de ejecución:** comparación entre el análisis secuencial y el paralelo.
- **Speedup:** mide la aceleración obtenida al ejecutar la versión paralela frente a la secuencial.
- **Eficiencia:** evalúa qué tan bien se utilizan los recursos del procesador en las ejecuciones paralelas.
- **Totales de eventos:** número de errores, advertencias e información encontrados en el archivo.

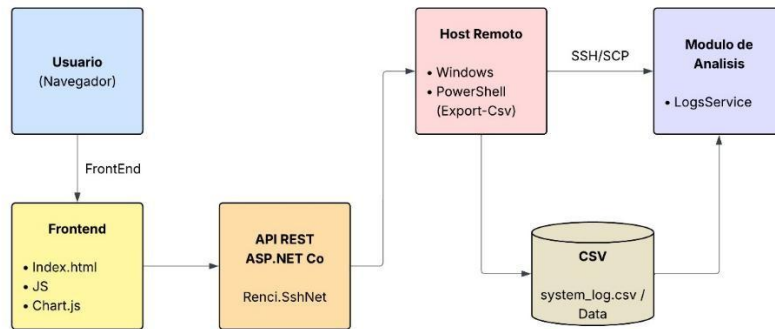
6. Aplicación a un problema del mundo real

Este sistema refleja un escenario real de **gestión y monitoreo de infraestructuras informáticas**, donde los administradores requieren obtener información confiable y rápida a partir de grandes volúmenes de logs. Al integrar la recopilación remota de datos y el análisis paralelo, el proyecto constituye una herramienta aplicable a la supervisión de servidores, la detección temprana de fallos y la auditoría preventiva en entornos empresariales y de misión crítica.

4. Diseño de la Solución

Arquitectura general del sistema

Diagrama General Del Sistema



La arquitectura del sistema está basada en un flujo cliente–servidor con un módulo de análisis paralelo. El proceso general es el siguiente:

1. Usuario (Navegador)

El usuario interactúa desde un navegador web, iniciando la petición de análisis de los logs.

2. Frontend

La interfaz está desarrollada con HTML, JavaScript y Chart.js, permitiendo tanto la interacción como la visualización de resultados mediante tablas y gráficos dinámicos.

3. API REST (ASP.NET Core)

La API recibe las solicitudes del frontend y gestiona la conexión remota hacia los dispositivos o servidores. Para la comunicación se utiliza la librería Renci.SshNet, que permite ejecutar comandos y transferir archivos vía SSH/SCP.

4. Host Remoto

El sistema se conecta a servidores Windows, donde se ejecutan comandos (por ejemplo, scripts de PowerShell con Export-Csv en Windows) que generan los archivos de logs requeridos.

5. CSV – Repositorio de datos

Los archivos generados se transfieren en formato CSV (Data/ system_log.csv), los cuales contienen los registros del sistema estructurados para su análisis.

6. Módulo de Análisis (LogsService)

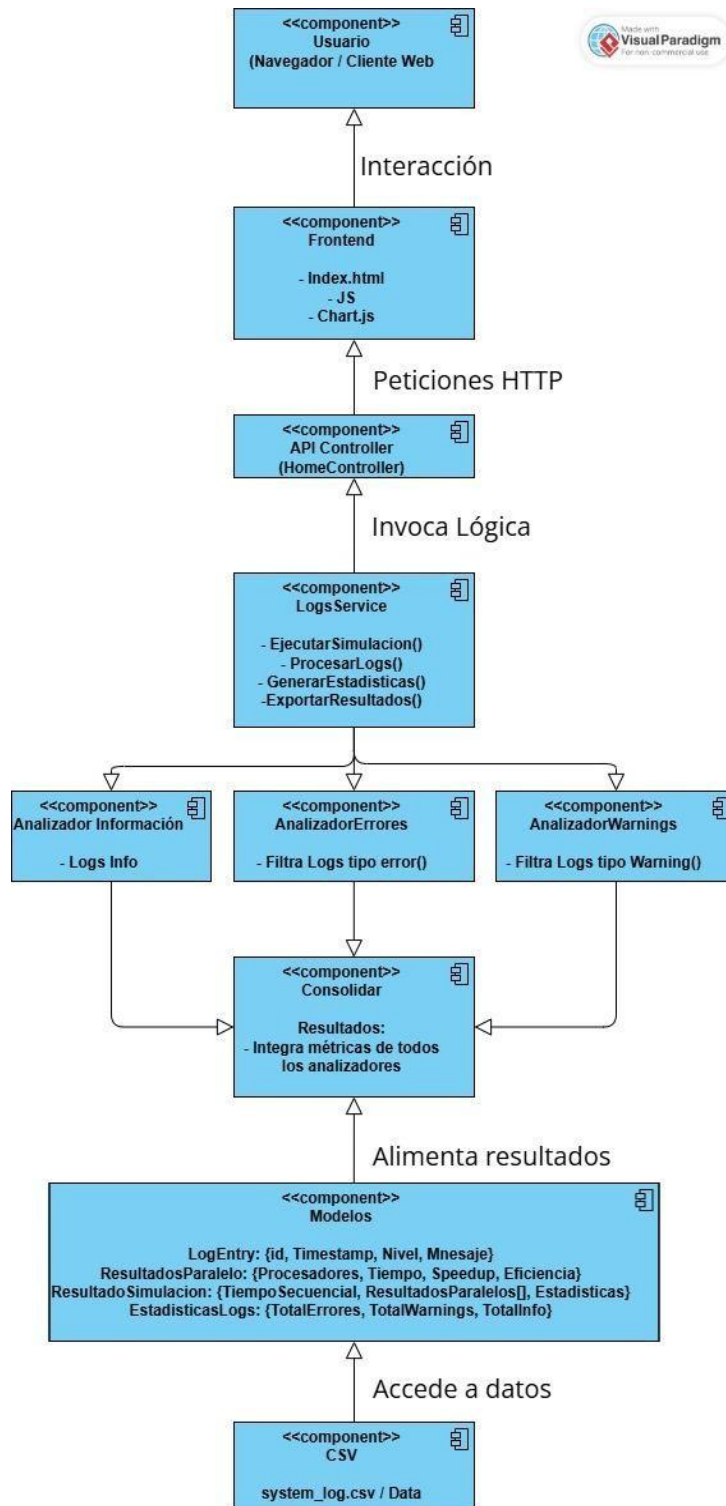
Una vez disponibles los logs en formato CSV, el módulo de análisis aplica procesamiento paralelo para identificar eventos relevantes (errores, advertencias, información general), calcular métricas y clasificar resultados.

7. Frontend (Visualización de Resultados)

Finalmente, los resultados del análisis se devuelven al frontend, donde se muestran métricas y gráficos que facilitan la interpretación del comportamiento del sistema.

Diagrama de componentes/tareas paralelas

El siguiente diagrama refleja los principales componentes y su interacción:



Estrategia de paralelización utilizada

El sistema implementa distintas técnicas de paralelización:

- **Descomposición de datos:** Los archivos de logs en formato CSV se dividen en bloques (líneas o secciones) que son procesados en paralelo. Cada bloque aplica la misma lógica de análisis (clasificación de errores, advertencias e información).

- Paralelismo de tareas: Además del análisis por bloques, diferentes hilos trabajan de forma simultánea en actividades como la lectura del archivo, conteo de eventos y generación de métricas, evitando cuellos de botella.
- Sincronización de recursos compartidos: Se emplean estructuras seguras y mecanismos de sincronización (locks, colecciones concurrentes) para consolidar los resultados parciales de cada hilo en un reporte global sin inconsistencias.
- Control del paralelismo: Se controla el grado máximo de paralelismo mediante parámetros de ejecución (por ejemplo, MaxDegreeOfParallelism), ajustando el número de hilos a los recursos disponibles del sistema.

Herramientas y tecnologías empleadas

- Lenguaje y Framework: C# con ASP.NET Core
- Comunicación remota: Librería Renci.SshNet para conexión SSH/SCP
- Procesamiento paralelo: Task Parallel Library (TPL), Parallel.ForEach, colecciones concurrentes
- Formato de datos: Archivos CSV para la transferencia de logs
- Frontend: HTML, JavaScript, Chart.js para la visualización de métricas
- Entorno de ejecución: Compatible con servidores Windows para la generación de logs

5. Descripción de la estructura del proyecto(Implementación Técnica)

El sistema tiene una estructura modular que sigue el patrón de organización en capas, dentro del proyecto se distinguen las siguientes carpetas y componentes principales:

Modelos (Models)

Son las clases que representan la información y resultados que se manejan en el sistema.

- **LogEntry:** Representa una entrada individual dentro del archivo de logs. Incluye atributos como:
 - **FechaHora:** Momento en que se generó el evento.
 - **Nivel:** Tipo de evento (Error, Advertencia, Información).
 - **Mensaje:** Contenido descriptivo del evento.
- **ParallelAnalysisResult:** Encapsula los resultados obtenidos tras el análisis de un archivo de log. Contiene:
 - **TotalErrors:** Cantidad de eventos clasificados como error.
 - **TotalInfos:** Número de mensajes informativos.
 - **TotalWarnings:** Cantidad de advertencias detectadas
 - **Métricas de rendimiento:** Incluye valores como tiempo de ejecución, speedup y eficiencia.
- **SSHConnectionDto:** Almacena los parámetros necesarios para establecer una conexión remota e información para los procesadores a utilizar, tales como IP, Puerto, Usuario, contraseña y Cores.

Servicios (Services)

Funcionan como intermediarios entre la lógica de negocio y la interfaz. Reciben las peticiones del usuario, invocan a los servicios correspondientes y devuelven los resultados del análisis.

- **ConexionRemotaService.cs:** Encargado de establecer la conexión con el servidor (Windows), ejecutar comandos o scripts, y transferir los archivos de log generados hacia el sistema local.
- **AnalisisLogsService.cs:** Implementa los métodos de análisis de logs tanto de manera secuencial como en paralelo. Entre sus funciones se encuentran:
 - **DividirArchivoEnBloques():** Segmenta un archivo de log en varias partes para permitir su procesamiento en paralelo.
 - **ProcesarBloque():** Lee y clasifica los eventos contenidos en un bloque.
 - **CombinarResultados():** Integra los resultados parciales de cada tarea paralela en un resultado global.

- **CalcularMetricas()**: Obtiene métricas como tiempo total, speedup y eficiencia.

Controladores (Controllers)

Concentran la lógica principal del sistema, tanto para la gestión de la conexión remota como para el procesamiento paralelo de los logs.

- **SSHController**: Gestiona el flujo completo desde la conexión remota, descarga de logs, ejecución del análisis secuencial/paralelo y retorno de los resultados al cliente.
- **environmentController**: Retorna los procesadores disponibles en la maquina host donde se esta ejecutando el API.

Explicación del código clave

El sistema se apoya en un conjunto de dos métodos centrales que permiten cumplir con los objetivos de conexión, procesamiento y análisis, dentro de estos métodos podemos encontrar abstracciones que utilicen otros métodos de clases distintas para realizar acciones, como es el caso de AnalyzeLogs:

- **GenerateAndDownloadCsv ()**: Se encarga de dividir el archivo en segmentos, facilitando que múltiples tareas puedan trabajar en paralelo sobre diferentes partes del mismo, reduciendo así el tiempo de análisis.
- **AnalyzeLogs()**: Se encarga de pasar la ruta del archivo de logs CSV previamente generado junto a los procesadores a utilizar para así devolver el análisis de logs secuencialmente como paralelamente en un objeto.

El método de AnalyzeLogs es un método que llama al servicio de Logs para ejecutar el método de **ProcesarLogsSecuencialVsParalelo** el cual hace la descomposición de datos para el análisis paralelo, utilizando los cores especificados a la par que también hace el análisis secuencial, este método se encarga de manipular la información del archivo CSV generado leyendo toda la información del mismo para retornar el análisis.

Uso de mecanismos de sincronización

El proyecto emplea técnicas para asegurar el acceso seguro a datos compartidos:

- **lock:** Aplicado en operaciones críticas donde es necesario evitar que dos hilos accedan simultáneamente a una misma variable de conteo, esto es utilizado para la segmentación de los logs en warning, error e info.

Implementación del frontend

El frontend del sistema se implementó como una **interfaz web hecha con HTML, CSS, JS**, que funciona como la capa de presentación encargada de mostrar al usuario los resultados del análisis de logs.

- **Interacción con el usuario:**
El frontend nos muestra un formulario con los datos requeridos para realizar el análisis host, puerto etc, y también nos muestra algunos tips para completarlo.
- **Comunicación con la lógica del sistema (backend):**
El frontend realiza 2 solicitudes HTTP hacia el backend, una para obtener los datos de cuanto procesadores tiene la maquina host y la otra para realizar el análisis de los logs.

Visualización de resultados:

La interfaz web muestra información clave del análisis de los logs:

- TiempoSecuencialMS
 - TiempoParaleloMS
 - Speedup y eficiencia
 - Cores utilizados en base a los disponibles en el host.
 - Cantidad de errores clasificados por tipo Warning,Info y Error.
- **Simplicidad y portabilidad:**
Al ser una interfaz Web la misma es ampliamente modular para desarrollar nuevas funcionalidades y agregar mas datos al dashboard de los análisis generado.

Justificación técnica de las decisiones tomadas

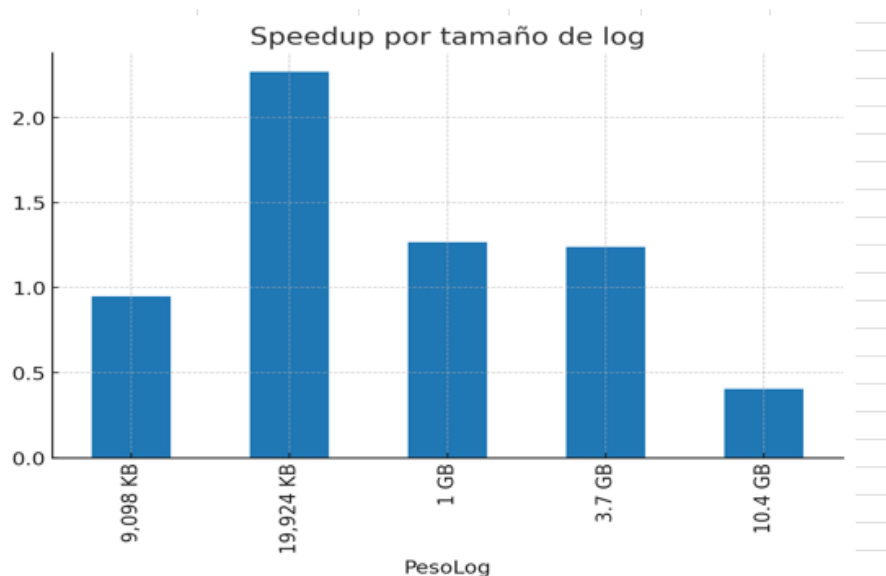
- **Uso de TPL (Task Parallel Library):** Se eligió por su facilidad de uso y soporte nativo en .NET para trabajar con paralelismo mediante estructuras como Parallel.ForEach.

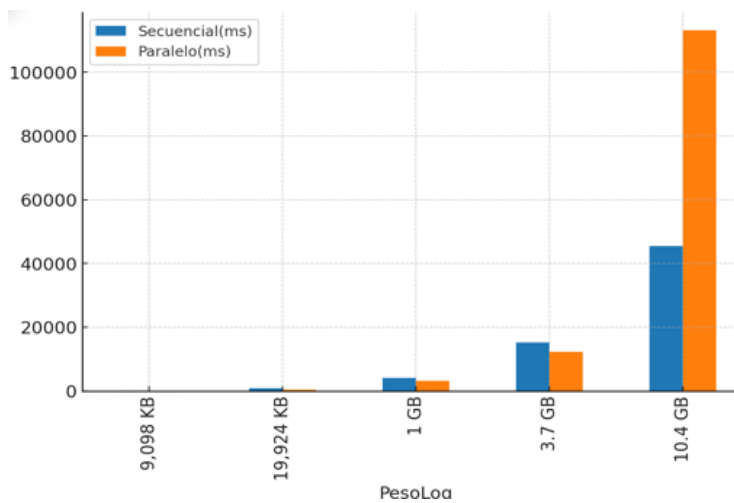
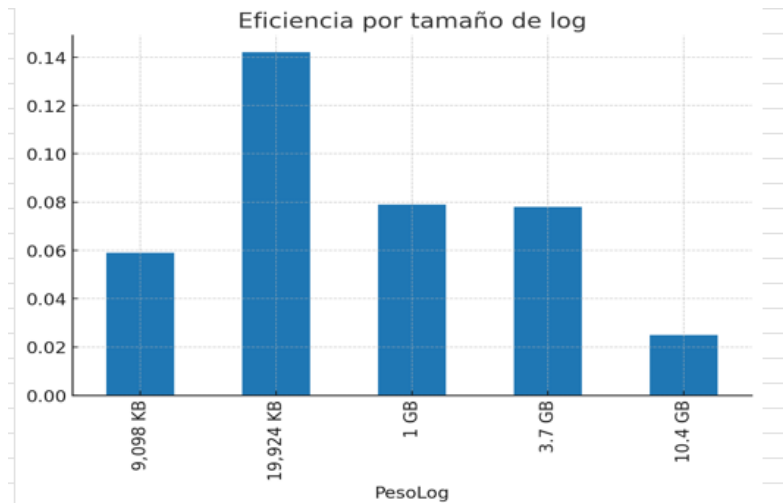
- **Separación en etapas:** Se dividió el análisis en fases (conexión remota, obtención de logs, procesamiento paralelo, combinación de resultados) para x mantener claridad y escalabilidad.
- **Estructuras concurrentes:** Su utilización minimiza riesgos de condiciones de carrera y simplifica la gestión de datos compartidos al utilizar **lock**.
- **Cálculo de métricas:** La inclusión de tiempos, speedup y eficiencia permite no solo validar el correcto funcionamiento del sistema, sino también evaluar la ganancia real obtenida al aplicar paralelismo.

6. Evaluación de Desempeño

Para la evaluación de desempeño utilizamos diferentes archivos de logs con diferencias en sus tamaños desde 9MB hasta 10GB, para probar el rendimiento del análisis para estos casos.

PesoLog	Errores	Warnings	Infos	Speedup	Eficiencia	Secuencial(ms)	Paralelo(ms)
9,098 KB	1625	1609	36331	0.948	0.059	73	77
19,924 KB	16449	16184	365456	2.267	0.142	884	390
1 GB	222328	661885	1325800	1.268	0.079	4073	3211
3.7 GB	786724	2356148	4707088	1.24	0.078	15172	12231
10.4 GB	2211400	6630994	13254524	0.403	0.025	45532	113101





7. Trabajo en Equipo

Descripción del reparto de tareas

† Frontend (interfaz y visualización de resultados):

Luis David Marte Vásquez (2023-1165)

† Backend (procesamiento paralelo, conexión remota y análisis de logs):

Luis Ángel Sánchez (2023-1681)

Jonathan José Frías Martínez (2023-1117)

† Documentación y coordinación general:

Gilberto Yunior Hernández De Los Santos (2023-1211)

Herramientas utilizadas para la coordinación

Para la gestión y organización del trabajo utilizamos **Git** y **GitHub**:

- ✦ **Git** permitió mantener un control de versiones distribuido, registrar cambios de forma segura y trabajar en paralelo sin conflictos mayores.
- ✦ **GitHub** funcionó como repositorio central en la nube, donde cada integrante subía sus aportes y se podían revisar mediante commits y merges para la unión de código de prueba a la rama funcional (main), lo que facilitó la colaboración y la integración continua.

8. Conclusiones

Principales aprendizajes técnicos

- ✦ **Luis David Marte Vásquez:** Trabaje en la construcción de la interfaz del sistema y la visualización de resultados en el Frontend. Implementé gráficos con Chart.js y aplique estilos generados manualmente para dar una presentación clara y organizada. Esto me permitió afianzar conocimientos en diseño de interfaces y integración de librerías gráficas para dashboards web dinámicos con js.
- ✦ **Luis Ángel Sánchez:** Trabajé en la segmentación de logs, la creación del dashboard HTML, la lógica en dashboard.js conectada al backend y la generación de métricas. Esto me permitió reforzar conocimientos en filtrado de datos, integración frontend-backend y representación clara de información en un dashboard web.
- ✦ **Jonathan José Frías Martínez:** Durante mi participación en el proyecto, uno de mis principales aprendizajes fue aplicar los conceptos de paralelismo que vimos en la teoría a un caso práctico. Trabajando en la parte del backend, entendí cómo estructurar métodos que simularan procesos en paralelo y cómo hacer que estos se integraran de forma ordenada con el resto de componentes del sistema. También aprendí la importancia de trabajar con datos compartidos entre hilos y cómo protegerlos mediante mecanismos de sincronización.
- ✦ **Gilberto Yuniór Hernández De Los Santos:** Como líder del equipo y encargado de la coordinación general, uno de mis principales aprendizajes fue cómo organizar el flujo de trabajo y mantener a todos los integrantes alineados en sus tareas. Pude entender mejor la importancia de planificar entregas parciales, dar seguimiento constante y asegurar que cada módulo del proyecto (frontend, backend y análisis) se conectara de manera correcta, también ayudando a agregar funcionalidades como el cálculo del speedup y la eficiencia.

Retos enfrentados y superados

- ✚ **Luis David Marte Vásquez:** Uno de los principales retos fue lograr realizar las validaciones y redirecciones al momento de obtener la información propuesta del backend. También fue un desafío encontrar un balance entre la estética y funcionalidad, lo cual resolví ajustando el CSS manualmente y validando la usabilidad con diferentes pruebas manuales de js.
- ✚ **Luis Ángel Sánchez:** Mi principal reto fue manejar la asincronía en la comunicación con el backend, ya que los datos no siempre se mostraban de forma consistente. También fue un desafío definir las métricas más relevantes, lo cual resolví con pruebas y ajustes en la lógica.
- ✚ **Jonathan José Frías Martínez:** Uno de los principales retos que enfrenté fue trasladar lo aprendido en clase a una aplicación web funcional. Al inicio me resultaba complicado entender cómo coordinar las tareas en paralelo sin que se produjeran errores en los resultados. Sin embargo, con práctica y apoyo del equipo, logré afianzar esos conceptos y adaptarlos al proyecto.
- ✚ **Gilberto Yuniór Hernández De Los Santos:** Uno de los principales retos fue lograr que todos los miembros trabajaran de forma sincronizada, ya que cada quien estaba enfocado en un área distinta. Coordinar los tiempos y unificar los avances en GitHub fue un desafío, pero nos permitió aprender a trabajar como un verdadero equipo de desarrollo.

9. Referencias

Fuentes bibliográficas, técnicas o académicas consultadas

1. Microsoft Docs – **Parallel Programming in .NET:**
<https://learn.microsoft.com/en-us/dotnet/standard/parallelprogramming/>
2. Microsoft Docs – **Task Parallel Library (TPL):**
<https://learn.microsoft.com/en-us/dotnet/standard/parallelprogramming/task-parallel-library-tpl>
3. Microsoft Docs – **Parallel.For and Parallel.ForEach in C#:**
<https://learn.microsoft.com/enus/dotnet/api/system.threading.tasks.parallel>
4. SSH.NET – **A Secure Shell (SSH) library for .NET:**
<https://github.com/sshnnet/SSH.NET>
5. Microsoft Docs – **Performance Considerations in Parallel Programming:**

<https://learn.microsoft.com/enus/dotnet/standard/parallel-programming/performance>

6. Powershell Get-WinEvent https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.diagnostics/get-winevent?view=powershell-7.5&utm_source=chatgpt.com
7. Export CSV https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/export-csv?view=powershell-7.5&utm_source=chatgpt.com
8. IEEE Xplore – **Speedup and Efficiency in Parallel Systems:** <https://ieeexplore.ieee.org/document/1435477>

10. Anexos

Manual del sistema

Abrimos un simbolo de terminal **CMD** para clonar el repositorio, dentro del mismo escribimos el siguiente comando:

- **git clone** <https://github.com/GilbertoHernandez150/Proyecto-Final-Paralela>

Esto nos clonará el repositorio de Github en nuestra maquina local para poder ejecutar el código del mismo.

```
C:\Users\lmarte\Desktop>git clone https://github.com/GilbertoHernandez150/Proyecto-Final-Paralela
Cloning into 'Proyecto-Final-Paralela'...
remote: Enumerating objects: 224, done.
remote: Counting objects: 100% (224/224), done.
remote: Compressing objects: 100% (97/97), done.
remote: Total 224 (delta 104), reused 220 (delta 102), pack-reused 0 (from 0)
Receiving objects: 100% (224/224), 196.04 KiB | 843.00 KiB/s, done.
Resolving deltas: 100% (104/104), done.
```

Una vez el repositorio este clonado necesitaremos de los siguientes requerimientos para poner cada módulo del proyecto a funcionar en conjunto. Que son los siguientes:

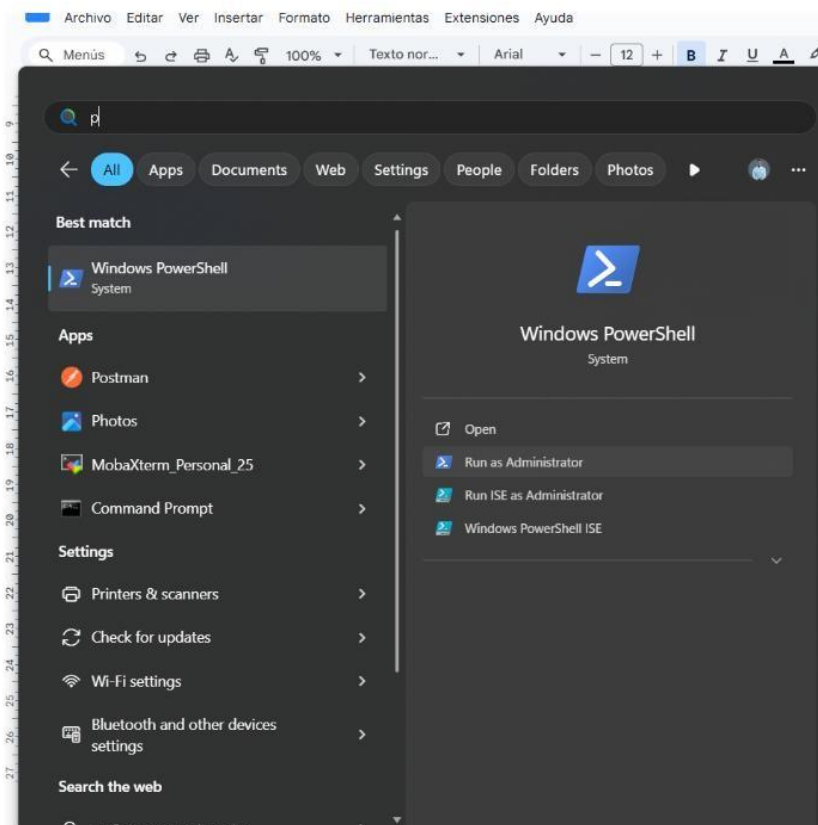
- Frontend: **Visual Studio Code** y Extension **“Live Server”**.
- Backend: Preferiblemente **Visual Studio** por simplicidad, pero también puede ser utilizado **Visual Studio Code**.
- Habilitar u instalar un servidor de Open SSH.
- Crear regla del firewall para abrir puerto 22 para el servicio de SSH.
- Un usuario local extra en la máquina host.
- Permisos de administrador del host donde se ejecutará el proyecto.

- Que el usuario a realizar la conexión ssh debe de ser agregado al grupo local de la máquina **“Log Event Readers”** o en español **“Lectores del registro de eventos”**.

Una vez listados estos requerimientos empecemos desde lo más complejo hasta lo mas sencillo, por lo que esta seria la división planteada para los pasos a seguir.

Configurar Open SSH en el host.

Hagamos la instalación del servidor de Open SSH por medio de **“Windows Powershell”**.



Le damos a que **si** para ejecutar la terminal como administrador y ejecutamos el siguiente comando, sustituimos le nombre-usuario por el usuario local al que nos conectaremos por **ssh**:

net localgroup "Event Log Readers" nombre-usuario /add

Dependiendo del idioma que tenga el sistema operativo este nombre de grupo va a variar, en el caso de no encontrar este grupo ejecutar el siguiente comando.

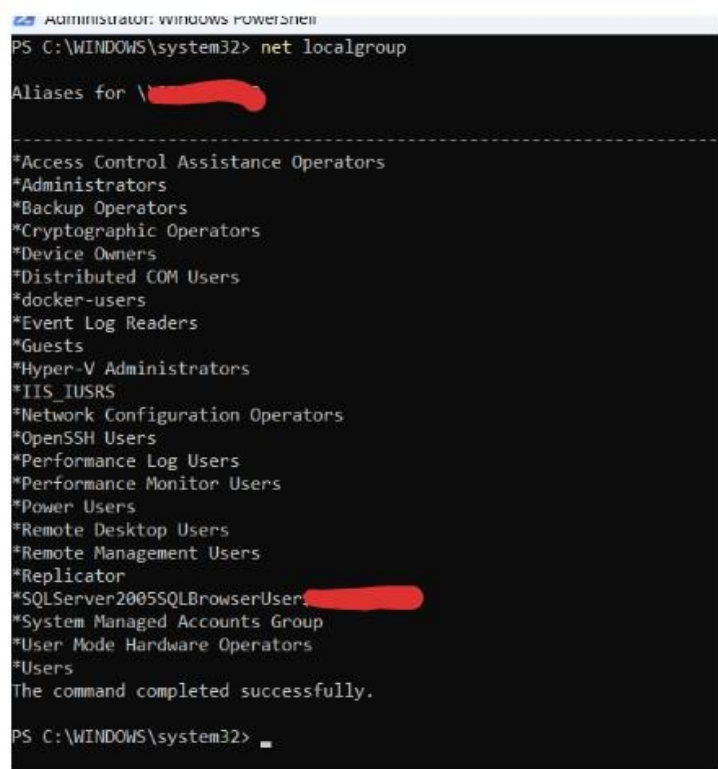
net localgroup "Lectores del registro de eventos" nombre-usuario /add

Output esperado:

```
The command completed successfully.  
net localgroup "Event Log Readers" itmanager /add  
>> C:\WINDOWS\system32>  
The command completed successfully.
```

En caso de tener algún error de que no encuentre el grupo, ejecuta el comando de ***net localgroup*** según los grupos listados busca similitudes en las traducciones de ***Event Log Readers*** o ***Lectores del registro de eventos***.

Output del comando:



```
Administrator: Windows PowerShell  
PS C:\WINDOWS\system32> net localgroup  
  
Aliases for \[redacted]  
  
-----  
*Access Control Assistance Operators  
*Administrators  
*Backup Operators  
*Cryptographic Operators  
*Device Owners  
*Distributed COM Users  
*docker-users  
*Event Log Readers  
*Guests  
*Hyper-V Administrators  
*IIS_IUSRS  
*Network Configuration Operators  
*OpenSSH Users  
*Performance Log Users  
*Performance Monitor Users  
*Power Users  
*Remote Desktop Users  
*Remote Management Users  
*Replicator  
*SQLServer2005SQLBrowserUsers [redacted]  
*System Managed Accounts Group  
*User Mode Hardware Operators  
*Users  
The command completed successfully.  
PS C:\WINDOWS\system32>
```

Ahora instalemos el servidor de “**Open SSH**”, para instalar este servicio de igual forma ejecutemos una terminal de powershell como administrador como se realizó anteriormente, dentro de esta ejecutemos el siguiente comando para habilitar o instalar este servicio de Windows.

Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0

Esto iniciará el proceso de instalación o configuración del servicio, una vez finalizado al volver a ejecutar el comando deberíamos de ver algunos de estos outputs:

output 1:

```
ore6
Add-WindowsCapability -Online -Name OpenSSH.Server~~~0.0.1.0

Path      :
Online    : True
RestartNeeded : False
```

output 2:

Name : OpenSSH.Client~~~~0.0.1.

State : Installed

Name : OpenSSH.Server~~~~0.0.1.

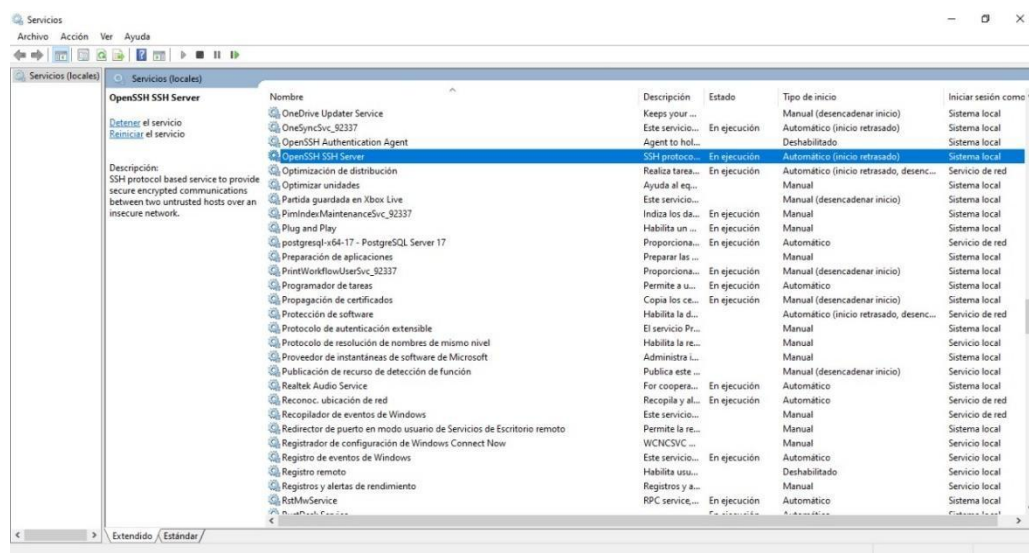
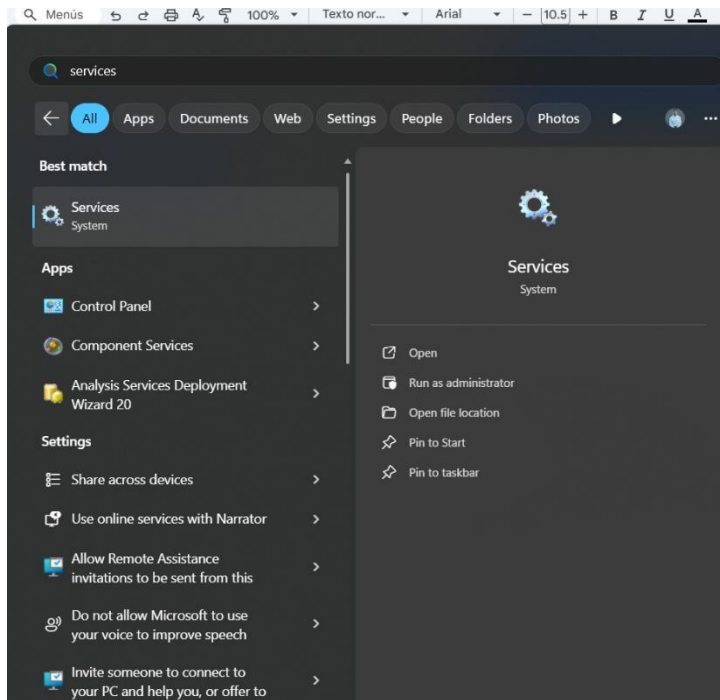
State : Installed

Una vez realizado esto debemos de habilitar una regla de firewall para abrir el puerto 22 para conexiones ssh, ejecutemos el siguiente comando:

Enable-NetFirewallRule -Name 'OpenSSH-Server-In-TCP'

Nota: esta es una regla predefinida en windows que esta deshabilitada, en el caso de que no la encuentres en tu sistema o no te funcione, crear una habilitando el puerto 22 de tu máquina.

Para validar que se haya instalado correctamente también y todo este funcionando vayamos a la parte de los servicios del sistema y busquemos por el servicio de **Open SSH Server**.



Si el servicio no está iniciado demos a **start..**

Hagamos una prueba rápida para verificar que la conexión funciona correctamente.

Abramos una terminal y ejecutemos el comando donde nombre-usuario es el usuario a que vas a querer realizar la conexión posterior.
`65+ ssh nombre-usuario@localhost`

Esto te mostrará un mensaje como el siguiente y debemos de aceptarlo escribiendo **yes.**

Esto es vital realizarlo si no, puede haber problemas al establecer la conexión y el sistema de fallos.

The authenticity of host 'example.com (192.0.2.1)' can't be established.

ED25519 key fingerprint is SHA256:xx.

Are you sure you want to continue connecting (yes/no/[fingerprint])?

Luego de aceptar esto, introduciremos la clave del usuario y ya realizamos la conexión exitosa.

Ahora procedamos a ejecutar el **Backend** con Visual Studio o Visual Studio Code.

Visual Studio Code:

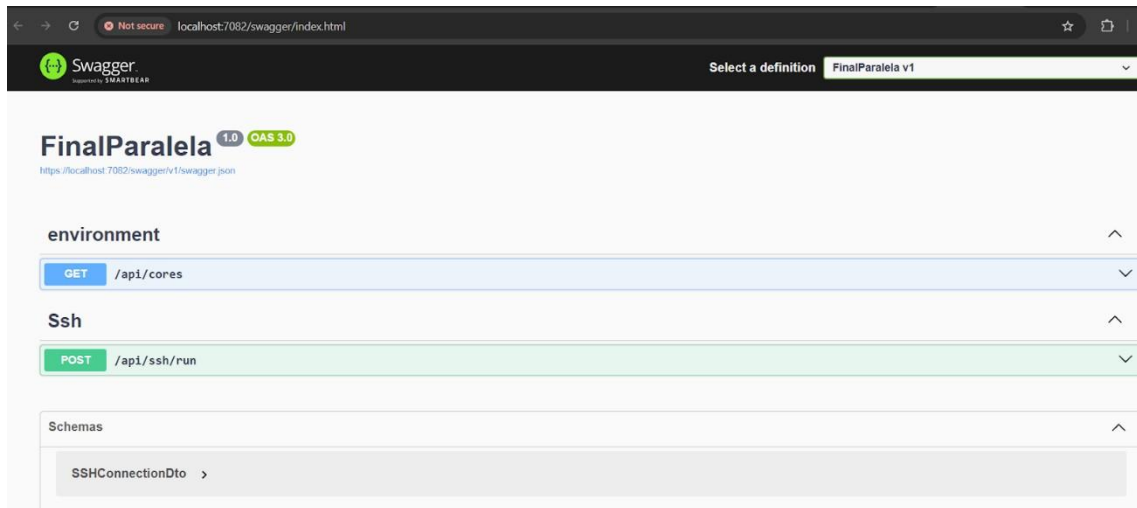
Abre una terminal y dirígete hacia donde tengas clonado el repositorio, una vez estes, dirígete a la siguiente ruta haciendo cd.

cd src/FinalParalela

Y lanzamos el backend colocando **dotnet watch run --launch-profile https**, si tenemos problemas con redirigir https, ejecutemos este comando para crear en los certificados autogenerados del proyecto para el localhost **dotnet dev-certs https --trust**. y volvamos a ejecutar el proyecto **dotnet watch run --launch-profile https**.

```
C:\Users\lmarte\Desktop\Proyecto-Final-Paralela\src\FinalParalela>dotnet watch run --launch-profile https
dotnet watch 🔥 Hot reload enabled. For a list of supported edits, see https://aka.ms/dotnet/hot-reload.
🔥 Press "Ctrl + R" to restart.
dotnet watch 🔧 Building C:\Users\lmarte\Desktop\Proyecto-Final-Paralela\src\FinalParalela\FinalParalela.csproj ...
dotnet watch 🏗️ Build succeeded: C:\Users\lmarte\Desktop\Proyecto-Final-Paralela\src\FinalParalela\FinalParalela.csproj
Using launch settings from C:\Users\lmarte\Desktop\Proyecto-Final-Paralela\src\FinalParalela\Properties\launchSettings.json
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7082
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5295
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\lmarte\Desktop\Proyecto-Final-Paralela\src\FinalParalela
```

Esto nos abrirá el navegador, mostrándonos los endpoints del backend.



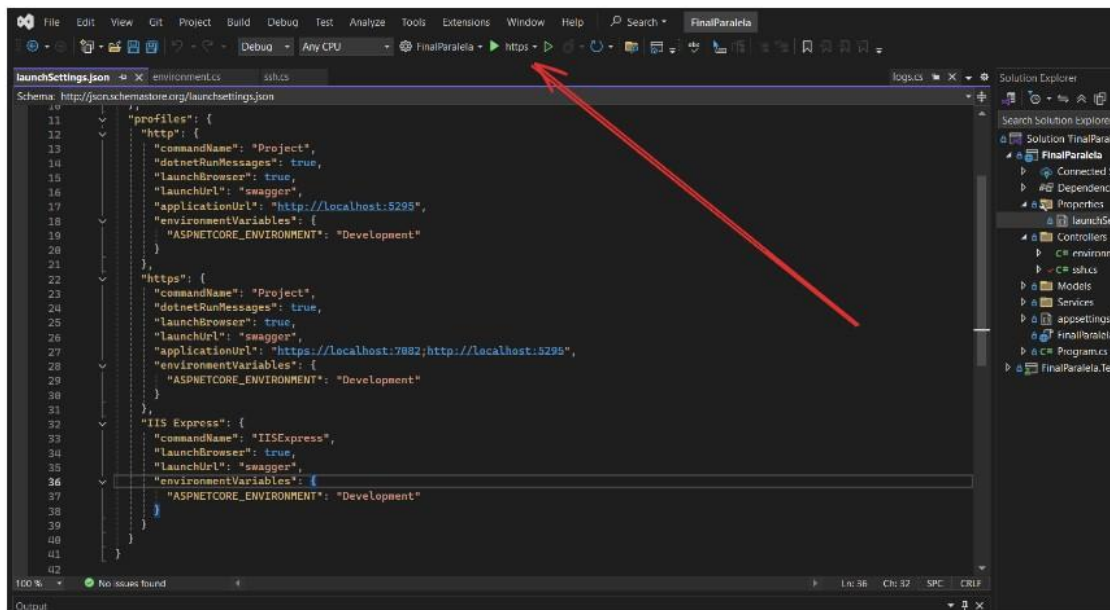
Nota: Es de vital importancia que el puerto 7082 este libre para correr la aplicación, en caso de que no cierre el proceso que lo este ejecutando o cambia la variable de **ASP_BACKEND_PORT** dentro de **src/frontend/js/index.js**, por el puerto donde este corriendo el API.

```
//Hacemos la solicitud con los datos para realizar el analisis
try {
  const ASP_BACKEND_PORT = 7082;
  //hacemos un await para no detener cualquier otra accion hasta obtener la info del analisis
  const response = await fetch(`https://localhost:${ASP_BACKEND_PORT}/api/ssh/run`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(formData) //parseamos la info a json para que se pueda reconocer en el b
  });

  //verificamos el estatus de la respuesta y mostramos el error
  if (!response.ok) {
```

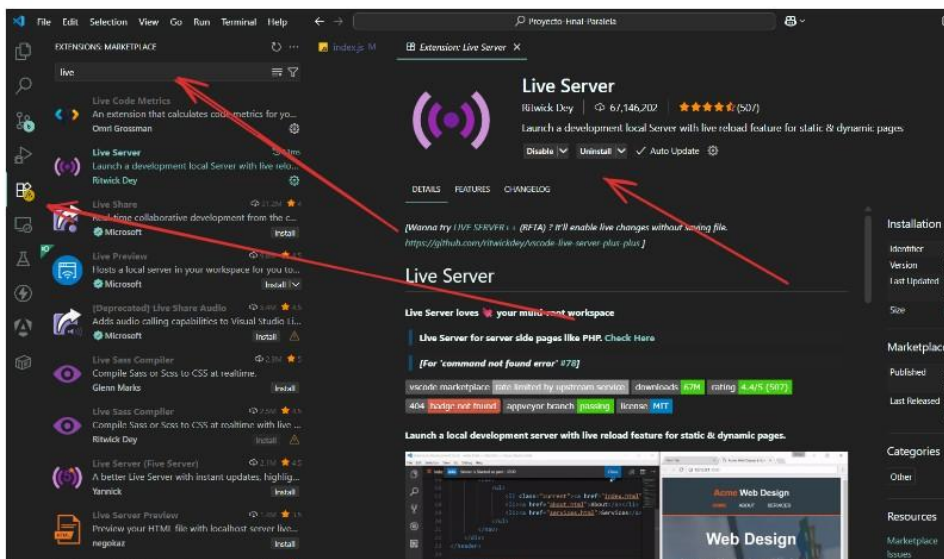
Visual Studio:

Hacemos click para ejecutar el proyecto por https, una vez lanzado se abrirá el navegador con el los endpoints del API como se hizo anteriormente con **Visual Studio Code**.

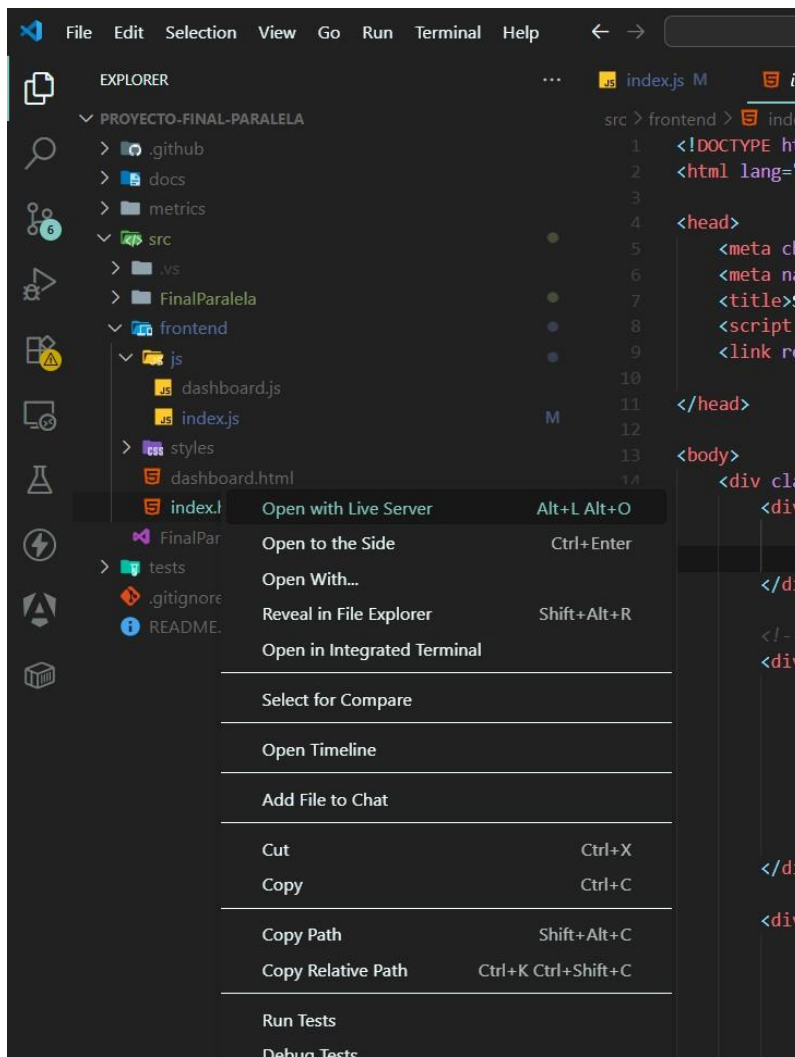


Nota: El proyecto solo admite solicitudes por https, por lo que si presentas algún problema al lanzar el backend, por favor da resolución a estos con algunas herramientas de debugging.

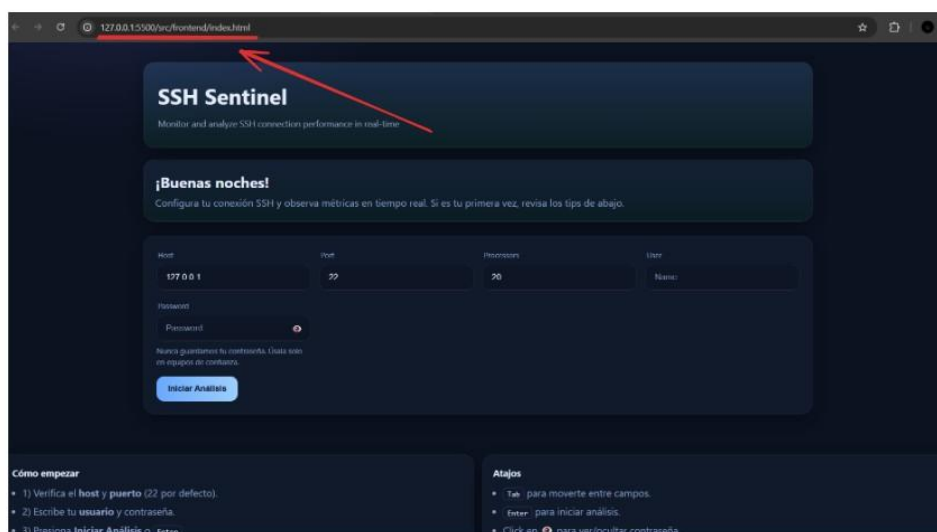
Ahora ya tenemos el backend listo, procederemos a ejecutar el Frontend, para ejecutarlo necesitaremos abrir **visual studio code** y verifiquemos en el apartado de las extensiones si tenemos **Live Server**, en el caso de no tenerla instalarla.



Nos dirigimos luego a los archivos del proyecto y hacemos click derecho sobre el `index.html` y corremos con la opción de **Open with Live Server**.



Esto nos lanzará un servidor web de manera automática sirviendo el contenido estático de la misma, debemos de procurar que la ruta donde haya sido ejecutado sea exactamente la misma que la de la imagen **/src/frontend/index.html**.



En caso de no ejecutarlo en esta ruta. dirígete a `src/frontend/js/`, dentro de esta carpeta hay 2 archivos de js, dentro de ambos hay una variable llamada **WEB_SERVER_URL** , esta cámbiala por el valor de tu servidor web.

`index.js`

```
setTimeout(()=>{
    // console.log("Te redireccionaremos al dashboard!!!");
    //hacemos la redireccion
    const WEB_SERVER_URL = "/src/frontend/";
    window.location.href = `${WEB_SERVER_URL}dashboard.html`;
    alert("Haremos la redireccion")
},1)
```

`dashboard.js`

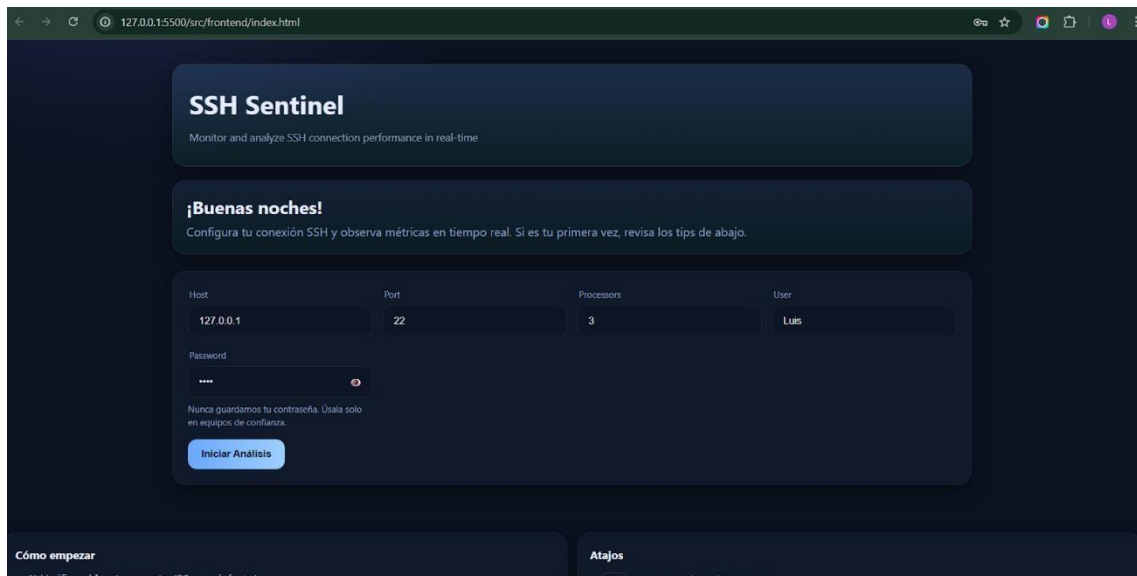
```
function getAnalysisData(){
    try{
        return JSON.parse(sessionStorage.getItem("analysis"));
    }
    catch{
        return null; //retornamos null ya que no hay ningun valor, podriamos retornar un
    }
}

//validaremos si tenemos la data del analisis
function validateAnalysisData(){
    const data = getAnalysisData();
    if(!data){
        const WEB_SERVER_URL = "/src/frontend/";
        alert("No pudimos obtener el analisis, te estaremos redireccionando al index nu");
        window.location.href = `${WEB_SERVER_URL}index.html`;
    };

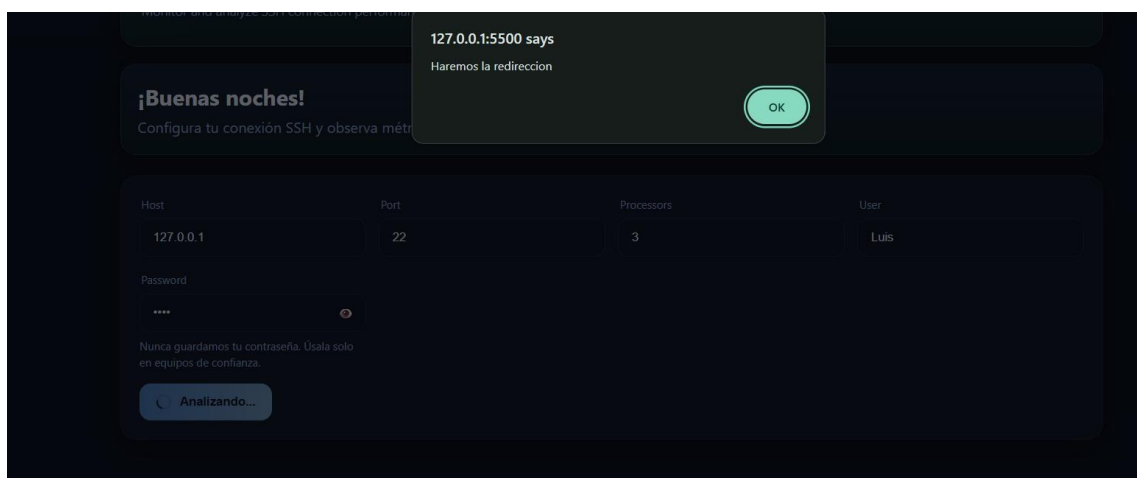
    //simulacion de que estamos cargando la informacion
    setTimeout(()=>{
        console.log("Te estaremos redireccionado en 5 segundos")

        //seteamos ahora el dashboard ya que si recibimos info
        const dashboard = document.getElementById("dashboard")
        dashboard.style.display = "block";

        // ocultamos el loading DESPUÉS de mostrar el dashboard
        const loading = document.getElementById("loadingOverlay");
        loading.style.display = "none";
    }, 3000)
```



En la pantalla inicial de SSH Sentinel, el usuario debe configurar su conexión SSH completando los siguientes campos: dirección del host (por ejemplo, 127.0.0.1), puerto de conexión (usualmente 22), cantidad de procesadores a utilizar para el análisis paralelo, nombre de usuario y contraseña. Una vez ingresados estos datos, se puede iniciar el análisis haciendo clic en el botón “Iniciar Análisis”. La interfaz también incluye recomendaciones de seguridad y tips para nuevos usuarios, facilitando una experiencia clara y guiada desde el primer uso.



Una vez que haces clic en “Iniciar Análisis”, el sistema comienza a procesar los logs en segundo plano utilizando los parámetros que configuraste (host, puerto, procesadores, usuario y contraseña). Mientras aparece el estado “Analizando...”, los datos se envían al

backend para ser clasificados y transformados en métricas. Cuando el análisis termina, el sistema redirige automáticamente al dashboard.



Una vez completado el análisis, el sistema redirige automáticamente al “Dashboard de Análisis Paralelo”, donde se muestran los resultados procesados en tiempo real. En esta vista, se destacan métricas clave como el total de errores (133), advertencias (323) e información general (4177), junto con indicadores de rendimiento como el speedup (4.625) y la eficiencia (1.15625). Además, se proyectan gráficas dinámicas: una de barras para comparar rendimiento y otra tipo donut para visualizar los tiempos de ejecución secuencial (37 ms) versus paralelo (8 ms). Esta interfaz permite interpretar rápidamente el impacto del procesamiento paralelo y facilita la toma de decisiones técnicas basadas en datos concretos.

Repositorio en GitHub: <https://github.com/GilbertoHernandez150/Proyecto-FinalParalela>