***Ecowatch Logs Application***

**Introducción**

Ecowatch es una herramienta de línea de comandos para procesar y analizar logs ambientales de distintas salas. El sistema simula la recepción continua de datos desde un archivo CSV, los valida, los almacena en una caché de ventana temporal, y genera reportes tanto en consola como en formatos tabulares (CSV/Excel).

**Arquitectura y Organización de Código**

ecowatch/ ← raíz del proyecto

├── src/ ← código fuente

│ ├── \_\_init\_\_.py

│ ├── config/ ← configuración y carga de variables de entorno

│ │ ├── settings.py ← clases de configuración (p.ej. rutas, umbrales)

│ │ └── \_\_init\_\_.py

│ │

│ ├── ingestion/ ← lógica de lectura y validación de logs

│ │ ├── \_\_init\_\_.py

│ │ ├── readers/ ← distintos formatos (.csv, .json, memoria)

│ │ │ ├── base.py ← ReaderInterface (SRP, DIP)

│ │ │ ├── csv\_reader.py ← implementación concreta

│ │ │ └── json\_reader.py

│ │ └── validators.py ← reglas de validación de cada registro

│ │

│ ├── domain/ ← modelos de las entidades del dominio

│ │ ├── \_\_init\_\_.py

│ │ ├── log.py ← clase Log (timestamp, tipo, sala, métricas)

│ │ ├── sensor.py ← clase Sensor

│ │ └── sala.py ← clase Sala

│ │

│ ├── cache/ ← caché temporal en memoria

│ │ ├── \_\_init\_\_.py

│ │ └── temporal\_cache.py ← TemporalCache (SRP: solo mantiene últimos 5′)

│ │

│ ├── reports/ ← generación de reportes

│ │ ├── \_\_init\_\_.py

│ │ ├── factory.py ← ReportFactory (Factory Pattern, OCP)

│ │ ├── base.py ← Report interface/abstract (ISP)

│ │ ├── status\_by\_sala.py ← ReportStatusBySala

│ │ ├── critical\_alerts.py ← ReportCriticalAlerts

│ │ └── pandas\_report.py ← Reporte utilizando librerias pandas

│ │

│ ├── services/ ← casos de uso y orquestación

│ │ ├── \_\_init\_\_.py

│ │ ├── ingest\_service.py ← coordina readers + validators + cache

│ │ └── report\_service.py ← invoca factory y ejecuta reportes

│ │

│ └── main.py ← punto de entrada (Orquesta los servicios)

│

├── tests/ ← pruebas unitarias (pytest/unittest)

│ ├── \_\_init\_\_.py

│ │

│ └── test\_reports.py

│

├── data/ ← datos, simulacion de fuente de datos

│ └── logs\_ambientales\_ecowatch.csv

│

├── exportación\_archivos/ ← datos de cache generado por el pandas\_report

│ ├── exportación\_CSV

│ │ └── pruebasexpo.csv ← datos de test

│ └── exportación\_EXCEL ← datos de cache generado por el pandas\_report

│

├── requirements.txt ← dependencias

├── .env ← variables sensibles

└── README.md ← overview del proyecto

\* `src/ingestion/`: lectores de logs y validadores.

\* `src/cache/`: implementación de caché deslizante (`TemporalCache`).

\* `src/domain/`: modelos de datos (`Log`, `Sensor`, `Sala`).

\* `src/services/`: servicios de ingestión y reporte.

\* `src/reports/`: reportes textuales y tabulares, con patrones Factory y Strategy.

\* `src/main.py`: interfaz CLI con Colorama y pandas.

Cada carpeta refleja una responsabilidad clara, facilitando mantenimiento y extensión.

**Elección de Estructuras de Datos**

\* \*\*`deque`\*\* para la caché temporal: permite inserción y remoción en los extremos en O(1), ideal para mantener una ventana de tiempo deslizante sin overhead de reconstrucción de listas.

\* \*\*`defaultdict(list)`\*\* en generación de reportes: agrupa logs por sala de forma natural y eficiente.

\* \*\*Lista de objetos `Log`\*\*: modelo fuertemente tipado para evitar errores de clave/atributo.

> \*Alternativa descartada:\* usar una base en memoria (Redis) habría agregado complejidad de infraestructuras y latencia de red, innecesaria para pruebas locales.

**Patrones de Diseño Implementados**

\* \*\*Factory (`ReportFactory`)\*\*: desacopla la creación de instancias de reporte de su uso, permitiendo añadir nuevos tipos sin modificar `main.py`.

\* \*\*Strategy (`MetricStrategy`)\*\*: encapsula algoritmos de métrica (avg, max, min) permitiendo extender cálculos (p. ej. desviación estándar) con nuevas clases sin tocar la lógica de `ReportStatusBySala`.

\* \*\*Decorator (propuesto)\*\*: ideal para logging, validaciones adicionales o benchmarking sin ensuciar la lógica de negocio. No se alcanzó a hacer

**Técnicas de Optimización**

\* \*\*Parsing de timestamps\*\* con `datetime.fromisoformat`: más rápido y sin dependencias externas que `dateutil`.

\* \*\*Simulación de streaming\*\* con `time.sleep(delay)` configurable: ofrece un modo rápido (delay=None) y otro realista para pruebas.

> \*Benchmarking:\* se puede envolver métodos clave con un decorator que mida tiempo de ejecución y registro de logs.

**Validaciones y Robustez**

\* \*\*`Validator.validate()`\*\*: transforma y verifica tipos (`datetime`, `float`) y valores (`estado` en `INFO|WARN|ERROR`), lanzando `ValueError` ante inconsistencias.

\* \*\*Decorators de validación\*\* (sugerido): garantizarían de forma declarativa la presencia de campos antes de procesar.

**Interfaz de Usuario y Experiencia**

\* \*\*CLI interactivo\*\* con Colorama: prompts coloreados, ejemplos inline, manejo de entradas inválidas.

\* \*\*Selección múltiple de reportes\*\*: ejecutar textuales y tabulares en un mismo ciclo.

\* \*\*Exportación automática\*\* a carpetas organizadas (`exportacion\_archivos/exportacion\_CSV` y `.../exportacion\_EXCEL`).

**Configuración y Escalabilidad**

\* \*\*`.env` + `python-dotenv`\*\*: mantiene rutas, retrasos y valores por defecto fuera del código, facilitando despliegues en distintos entornos (dev, staging, prod).

\* \*\*`Settings`\*\* centraliza la carga de variables de entorno, mejorando la portabilidad.

**Tests**

\* \*\*test\_reports.py\*\*: Feed un pequeño conjunto de Log a ReportStatusBySala y ReportCriticalAlerts, y comprueba que los resultados (dicts y listas) tienen los valores esperados.

**Comparación de Alternativas**

| Componente | Alternativa | Nuestra elección | Justificación |

| ----------------- | --------------- | ---------------------------------------- | -------------------------------- |

| Caché temporal | Redis in-memory | `deque` en Python puro | Ligero, sin dependencia externa |

| Reporting tabular | Jinja2/HMTL | `pandas.DataFrame` + `to\_csv`/`to\_excel` | API rica, familiar para análisis |

| Validaciones | Manual `if` | `Validator` + decoradores sugeridos | Separación clara y reutilizable |

**Futuros Trabajos**

\* Desplegar un \*\*API REST\*\* (FastAPI) para servir reportes en JSON.

\* Integrar \*\*monitoring\*\* y métricas de uso con decorators de benchmarking.

\* Añadir \*\*nuevos formatos de entrada\*\* (JSON, MQ) implementando `ReaderInterface`.

\* Expandir patrones (Decorator para caching extendido, Strategy para filtros de alertas).

***Instalación y Puesta en Marcha***

Sigue estos pasos para preparar y ejecutar la aplicación, así como su suite de pruebas:

1. \*\*Clonar el repositorio\*\*

```bash

git clone <URL-del-repositorio>

cd ecowatch

```

2. \*\*Crear y activar un entorno virtual\*\*

- Windows PowerShell:

```powershell

python -m venv venv

.\venv\Scripts\Activate.ps1

```

- Unix/macOS:

```bash

python3 -m venv venv

source venv/bin/activate

```

3. \*\*Instalar dependencias\*\*

```bash

pip install -r requirements.txt

```

Tu `requirements.txt` debe incluir al menos:

- colorama

- python-dotenv

- pandas

- pytest

- pytest-cov

4. \*\*Configurar variables de entorno\*\*

Crea un archivo `.env` en la raíz con:

```dotenv

LOG\_CSV\_PATH=./data/logs\_ambientales\_ecowatch.csv

STREAM\_DELAY\_SEC=

DEFAULT\_REF\_TIME=2025-05-01T08:05:00

DEFAULT\_WINDOW\_MIN=5

```

Ajusta rutas y valores según tu entorno.

5. \*\*Ejecutar la aplicación\*\*

```bash

python -m src.main

```

- Sigue las indicaciones del menú: ingresa hora de referencia, ventana en minutos, salas y tipos de reporte.

- Para exportar, elige la opción CSV o Excel; los archivos quedarán en `exportacion\_archivos/exportacion\_CSV/` o `.../exportacion\_EXCEL/`.

6. \*\*Ejecutar pruebas\*\*

- \*\*pytest\*\* (test\_ingest):

```bash

# Asegúrate de incluir 'src' en PYTHONPATH:

export PYTHONPATH=$(pwd)/src # Unix/macOS

set PYTHONPATH=%CD%\src # Windows PowerShell

```

- \*\*unittest\*\* (test\_reports):

```bash

python -m unittest tests/test\_reports.py

```

- \*\*Cobertura\*\*:

```bash

pytest --cov=src tests/

```

---

\*Esta documentación describe las decisiones de diseño, estructuras de datos elegidas, patrones aplicados y opciones de escalabilidad para el sistema Ecowatch Logs.\*