# Documentación Técnica - Sistema de Monitoreo EcoWatch

## Descripción General

El sistema desarrollado permite la gestión eficiente de logs ambientales en tiempo real, su almacenamiento temporal en memoria, y la generación de reportes ejecutivos adaptables para la toma de decisiones. Está diseñado para ser modular, extensible y fácil de mantener.

Se utilizaron estructuras de datos adecuadas, se aplicaron patrones de diseño (Factory y Strategy) para desacoplar responsabilidades, y se priorizó la claridad y escalabilidad del código.

## Descripción general del sistema

EcoWatch es un sistema de monitoreo ambiental que permite:

* Leer logs ambientales desde archivos CSV.
* Validarlos y almacenarlos en una base de datos relacional (MySQL).
* Mantener en memoria los últimos logs utilizando un sistema de caché temporal.
* Simular el avance del tiempo para evaluar el comportamiento del sistema en tiempo real.
* Generar reportes personalizados de forma modular.

Su arquitectura fue diseñada para ser extensible y facilitar futuras modificaciones.

## Arquitectura general y flujo de datos

### Carga de logs:

Se realiza desde un archivo CSV. Los datos se validan y se insertan en la base de datos.

El csv se lee desde la funcion cargar\_logs\_csv(ruta) del archivo log\_loader.py. Allí se valida que que tenga todos los campos esperados (EXPECTED\_FIELDS) que son: "timestamp", "sala", "estado", "temperatura", "humedad", "co2", "mensaje".  
Se parsean y validan los datos para que python pueda manejarlos sin problema. Los logs validos se van guardando en un array de log\_validos, y si hubiese no validos en un array de errores.   
Dentro del array de log\_validos se va creando la estructura que luego va a guardarse en la base de datos. Por un lado una instancia de **LogRaw** clase que mapea a la base de datos y la sala.

**Justificación:**

En esta iteración decidí leer solo archivos .csv y priorizar la persistencia de los logs en la base de datos.

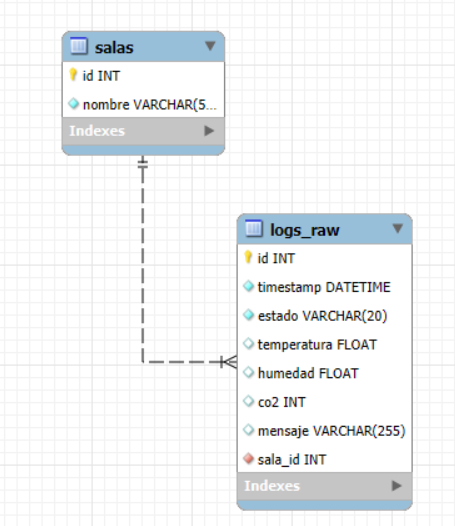
### Base de datos

Utiliza SQL Alchemy como ORM y MySQL como Sistema de gestión de base de datos.  
La estructura armada en el item anterior (carga de logs) se itera y se va guardando en la base de datos.

**Modelos:** Para la persistencia en la base de datos elegí dos modelos:

* “Sala” que modela la información de referente a la Sala donde realizan las mediiciones y por otro lado
* “LogRaw” donde se modela la información de los logs.
* Relación: Entre ambas clases es: 1 a N: Una Sala puede tener muchos logs.

**Base de datos:** El ORM mapea las clases y se guardan en las tablas: salas y logs\_raw



Para el guardado en la base de datos se eligió utilizar session que proporciona el ORM e ir iterando sobre los logs leídos desde el .csv y luego al final de la lectura recién se hace un commit. Si existe algun problema en la carga se hace un rollback.

**Justificación:**

Elegí realizar la persistencia en MySQL porque es un motor de base de datos muy robusto y facil de utilizar y configurar.  
SQLAlchemy lo elegí tambien por su facilidad de uso y porque en esta instancia al no desarrollar una API, debía tener un ORM despegado de frameworks de backend.

**Decisión de diseño:**

Decidí que los logs primero se carguen la base de datos y luego en la memoria, porque supuse que de los sensores ubicados en las diferentes salas, llegan mediciones y se guardan en la base de datos para protección de los mismos, y luego se levanta en cache para realizar análisis. Elegí este flujo de datos pensando en que si se cargan primero en cache para luego ir a base de datos, puede suceder algo y perder información.

Los modelos elegidos fueron simplemente sala y logs, descartando sensores, ya que sobre los mismos no existe información.

### Caché Temporal:

Para el uso del cache temporal utilice dos clases principales un DepuradorLogs y Un CacheTemporalLogs:

#### CacheTemporalLogs:

Implementa un **SortedDic** para tener los logs en memoria ordenados por timestamp.

Tiene las siguientes funciones:

* logs\_desde\_db(): (Metodo de clase) que llama a la función actualizar\_desde\_db(): En esta se crea una session para leer los logs desde la base de datos y filtra solo los que estan en el umbral de 5 minutos. Osea desde 5 minutos atrás del valor de **ahora** y el valor de **ahora**.
  + Lo que se lee de la base de datos se guarda en un modelo de datos LogEntry creado especificamente para guardar los logs que se utilizan en el caché. Este modelo no tiene relaciones simplemente tiene la información de forma lineal.
  + Luego se guarda esta estructura en el cache a traves de la función agregar\_log()
* agregar\_log(): agrega un nuevo log en el cache, registra el timestamp en el depurador y agrega el log en el SortedDic.
* obtener\_logs(): Obtiene todos los logs en el rango de fechas especificados. Retorna desde el cache una lista de LogEntry.
* obtener\_todos(): Obtiene todos los logs en cache.
* depurar(): elimina los logs antiguos basándose en la ventana temporal de tiempo. Actualmente esta funcionalidad solo se puede probar simulando el cambio de la variable **ahora** para que pueda verse como se corre la ventana.
* simular\_tiempo: Como no existen logs que van llegando la unica forma de simular tiempo es forzando una acción. En esta función se va corriendo la ventana de cache X cantidad de minutos y se puede ver como se van eliminando los logs viejos. Por otro lado cuando la ventana va corriendo y ya no hay mas logs, se vuelven a leer desde la base de datos para mantener los datos actualizados en cache, pero solo por el tiempo faltante (variable **actual**).

**Decisión de diseño:**

Implementé una variable **ahora** que puede ser cargada manualmente en el programa para establecer cual es el momento desde el cual se va a comenzar a cargar los datos en caché. Esto debido a que los datos del .csv son todos del 1/05/2025 entre las 8 y las 10 de la mañana. Esta variable se puede cambiar al momento actual cuando el sistema reciba logs en tiempo real.

#### DepuradorlLogs:

Asiste a la clase CacheTemporalLogs y se encarga de mantener actualizada la memoria cache. Su responsabilidad principal es depurar los logs que hayan quedado fuera de una ventana temporal que en este caso son 5 minutos.

Tiene las siguientes funciones:

* registrar\_timestamp(): Cada vez que se agrega un log nuevo al sistema, se registra su timestamp en una estructura interna (deque) que permite mantener un orden cronológico.
* depurar\_logs(): Se encarga de calcular el límite inferior de la ventana, eliminar del cache todos los logs que tengan un timestamp anterior a ese límite y elimina los timestamps de la deque interna si ya están fuera del rango.
* obtener\_umbral\_actual: devuelve el umbral actual calculado con base en el timestamp más reciente
* limpiar\_todo(): Limpia completamente los timestamps registrados

**Decisión de diseño:**

Internamente usa una **deque** que permite eliminar elementos usando FIFO y SortedDic, que son los datos del cache.

Por el principio de responsabilidad unica se desglosa el manejo de cache en dos clases, CacheTemporal y Depurador.

### Reportes

El sistema de reportes permite al usuario seleccionar el tipo de reporte a generar desde un menú por consola. Actualmente, los reportes se basan en los logs que se encuentran en memoria (cache), que ya fueron previamente leídos desde la base de datos.

Se implementaron los siguientes tipos de reportes:

* Promedio por métrica: calcula el promedio de temperatura, humedad y CO2 por sala.
* Alertas críticas: contabiliza la cantidad de estados críticos como WARNING o ERROR por sala.

#### Patrones de Diseño utilizados:

**Strategy:**

Cada tipo de reporte implementa su propia estrategia de cálculo a través de una clase que hereda de una interfaz base (Reporte). Esto permite encapsular la lógica de cada tipo de reporte en una clase independiente, facilitando el mantenimiento y la extensión.

* PromedioPorMetricaReport: genera un dataframe con el promedio de una métrica específica agrupada por sala. Recibe cualquier metrica.
* AlertasCriticasReports: filtra los logs por un tipo de estado (ej: "WARNING") y cuenta cuántas veces aparece por sala. Recibe cualquier estado.
* En ambas implementaciones utilice DataFrame de Pandas, para convertir el cache de logs a un dataframe para poder realizar operaciones de agrupamiento, filtrado y agregación.

**Decisiones de Diseño:**

Decidí crear un tipo de patrón strategy para que se pueda armar los reportes como un rompecabezas. La primera estrategia, promedio por metrica, dependiendo de lo que quiera el usuario se puede crear un reporte por temperatura, humedad o co2. Esta pensado para que luego pueda armarse uno con todas o parte de las metricas en la implementación del patrón factory. La segunda estrategia, alertas criticas, tambien esta pensado para armarse según la necesidad del usuario, si solo necesita “warnings” o “error” y otra en el futuro.

**Factory:**

Utilicé una clase **ReporteFactory** que permite al usuario seleccionar dinámicamente el tipo de reporte que desea ejecutar. Esta clase actúa como una factoría que devuelve la instancia correspondiente según el tipo de reporte solicitado.

Las instanciaciones de la Factoria son:

* PromedioPorMetricasVarias: permite generar múltiples reportes de promedio de forma dinámica (por temperatura, humedad, CO2).
* AlertasCriticasVarias: permite contar múltiples tipos de alerta (por ejemplo WARNING y ERROR).
* En estas clases utilice ambos patrones para ejecutar varias estrategias sin modificar el sistema base de reportes.

**Ejemplo de Uso:**

reporte = ReporteFactory.elegir\_reporte("promedio")

print(reporte.generar\_reporte(df, "temperatura", "humedad", "co2"))

reporte = ReporteFactory.elegir\_reporte("alertas")

print(reporte.generar\_reporte(df, "WARNING", "ERROR"))

pero tambien se podria crear un reporte solo de temperatura:

reporte = ReporteFactory.elegir\_reporte("promedio")

print(reporte.generar\_reporte(df, "temperatura"))

o solo de “WARNING”:

reporte = ReporteFactory.elegir\_reporte("alertas")

print(reporte.generar\_reporte(df, "WARNING"))

**Decisiones de diseño:**

Elegí Factory para que el usuario pueda elegir entre los reportes y porque luego puedo agregar mas tipos de reporte. Sumado a Strategy, ambos me permiten la encapsulación de la lógica de cada reporte y la reusabilidad.

**Decorator (mención adicional)**

Aunque no implementé un @decorator clásico en términos sintácticos, se apliqué el patrón Decorator de manera conceptual en la función auxiliar pausar() que fue utilizada como decoración de salida para los reportes. Esta función permite mejorar la experiencia de usuario sin modificar la lógica interna de generación del reporte, encapsulando la pausa y limpieza de consola. Podría modularizarse aún más con functools.wraps si se necesitara reutilizar como decorador real.

## Estructura de datos utilizadas:

* SortedDict (de sortedcontainers): permite mantener los logs en memoria ordenados por timestamp, lo que optimiza las búsquedas por rango temporal.
* deque (de collections): usada en DepuradorLogs para mantener los timestamps ordenados y permitir depuración eficiente por ventana de tiempo (FIFO).
* DataFrame (de pandas): ideal para generar reportes flexibles mediante agrupamientos, agregaciones y filtrados de columnas.

## Conclusiones

La arquitectura propuesta permite cumplir con los requisitos funcionales de EcoWatch, facilitando la extensibilidad y mantenimiento del sistema en el futuro. Las decisiones de diseño las tomé priorizando la separación de responsabilidades, la eficiencia del sistema y la claridad del código.

Gracias al uso de patrones de diseño como Strategy y Factory, la lógica de generación de reportes es fácilmente escalable y adaptable a nuevos requerimientos.

## Consideraciones sobre funcionalidades futuras

Este sistema fue diseñado con una arquitectura extensible que permite incorporar fácilmente nuevas funcionalidades, como:

* Generación de reportes personalizados a partir de múltiples métricas combinadas
* Simulación avanzada del flujo de tiempo en el sistema con logs en tiempo real o generados automáticamente.

Si bien estas capacidades fueron contempladas en el diseño general, las incluí en la implementación actual debido a limitaciones de tiempo. prioricé entregar una base sólida, clara y funcional sobre la cual puedan agregarse dichas extensiones sin necesidad de reescribir la lógica existente.