DOCUMENTACIÓN HOMEWORK ENTREGABLE - HENRY

## Introducción al Proyecto

El siguiente documento tiene por objetivo ahondar más a fondo en el entendimiento de las clases del homework entregable del proyecto de registro de logs ambientales. Todo lo referido a el proyecto en sí, como instalarlo, que dependencias utiliza, como consultar los endpoints y demas, por favor lea el readme del [repositorio](https://github.com/garciaMatiasNicolas/Henry-HomeworkEntregable)

Este proyecto se construyó como una api rest con tres endpoints principales:

1. POST para ejecutar la carga de archivo desde los datos csv
2. GET de un reporte de los logs agrupados por sala en un periodo de tiempo
3. GET de un reporte de los logs críticos en un periodo de tiempo

### Funcionamiento General

La idea base del proyecto fue construir una clase de tipo **Caché** que almacena en memoria aquellos logs cuyo timestamp sea **dentro de los últimos 5 minutos** desde el momento de ejecución del POST.  
 Aquellos logs más antiguos (más de 5 minutos) son directamente insertados en una **base de datos MySQL** persistente.

El procesamiento de archivos, el guardado inteligente de los logs, y la generación de reportes son manejados a través de distintas clases organizadas en módulos.

## Estructura de carpetas

Dentro de la carpeta src del proyecto, se encuentra el archivo **main.py**, encargado de la configuración del servidor y el ruteo de los endpoints mediante **Fast API** y **uvicorn**. Luego hay tres carpetas principales:

* **model**: Dentro de esta carpeta esta el modelo de los Logs que van a ser almacenados tanto en caché como en la base de datos
* **services:** Contiene todas las clases que manejan lógica de negocio, carga de datos, estrategias de reportes, acceso a base de datos, caché y ejecución
* **app**: Dentro de esta carpeta esta todo lo relacionado a la api, tanto los controladores de los endpoints como así también la ruta de los mismos
* **data** (fuera de src): Aqui tiene que estar nuestro archivo de logs: ***logs\_ambientales\_ecowatch.csv***

## Detalle de clases

## *Class Logs*

* **Ubicación**: src/model
* **Objetivo**: definir el modelo de los logs que se esperan para almacenar tanto en base de datos como en caché.
* **Atributos**:
  + **Status**: String
  + **Message**: String
  + **Humedity**: float
  + **Sala**: string
  + **Temperature**: float
  + **Timestamp**: datetime
  + **Co2**: int
* **Métodos:**
  + **init:** Instancia la clase con los atributos pasados por parámetros
  + **from\_db\_row**: Transforma una tupla de base de datos a un Log
  + **form\_row**: Transforma de una row de pandas a un Log

## *Class Cache*

* **Ubicación**: src/services
* **Objetivo**: definir el modelo del almacenamiento de caché y los métodos que lo componen.
* **Atributos**:
  + **logs**: Deque de Logs (mantiene los logs en orden de llegada para eliminar los viejos)
  + **by\_sala**: Dict de listas de logs cuya key son las Salas
  + **by\_timestamp**: Dict de listas de logs cuya key son timestamps
* **Métodos:**
  + **init:** Inicializa la estructura de caché, preparando las colecciones vacías necesarias para el almacenamiento y acceso eficiente a los logs.
  + **add\_log**: Agrega un nuevo log y elimina los que tienen más de 5 minutos.
  + **get\_logs\_by\_sala**: Devuelve los logs asociados a una sala, eliminando antes los expirados.
  + **get\_log\_by\_timestamp**: Devuelve el log correspondiente al timestamp dado, si existe.
  + **\_clean\_old\_logs**: Elimina logs con más de 5 minutos de antigüedad para mantener la caché limpia.

*Class MYSQLDataBase*

* **Ubicación**: src/services
* **Descripción**: Administra la conexión a una base de datos MySQL y permite crear tablas e insertar logs.
* **Atributos**:
  + **CONN**: Conexión activa a la base de datos MySQL.
  + **CURSOR**: Cursor para ejecutar comandos SQL.
  + **CREATE\_TABLE\_QUERY**: Query SQL para crear la tabla templogs si no existe.
* **Métodos**:
  + **create\_tables**: Ejecuta la query para crear la tabla templogs si no existe y guarda los cambios.
  + **insert\_logs**: Insertar múltiples registros de logs en la base de datos a partir de una lista de objetos Logs.

*Class DataLoaderFactory*

* **Ubicación**: src/services
* **Descripción**: Crea dinámicamente un loader según el tipo de fuente (csv o json) y valida que los datos tengan las columnas requeridas.
* **Atributos**:
  + **source**: tipo de archivo (csv o json)
  + **filepath**: ruta del archivo
* **Métodos**:
  + **create\_loader**: Instancia el loader adecuado.
  + **validate**: Carga los datos y verifica que las columnas sean exactamente las esperadas.

*Class ReportStrategy (abstracta)*

* **Ubicación**: src/services
* **Descripción**: Define la interfaz para estrategias de generación de reportes.
* **Método** generate(data): Debe ser implementado por cada estrategia para generar un reporte a partir de un DataFrame.

*Class StatusSalaReportStrategy*

* **Ubicación**: src/services
* **Descripción**: Genera un resumen por sala con promedios de temperatura, humedad y CO2.
* **Método** generate(data): Devuelve un diccionario con los promedios agrupados por sala.

*Class CriticalAlertsReportStrategy*

* **Ubicación**: src/services
* **Descripción**: Detecta registros críticos según niveles altos de CO2, temperatura o baja humedad.
* **Método** generate(data): Devuelve un diccionario con los registros críticos.

*Class ReportFactory*

* **Ubicación**: src/services
* **Descripción**: Devuelve la estrategia de reporte adecuada según el nombre indicado.
* **Método** estático get\_report(name): Retorna una instancia de la estrategia solicitada (status\_sala o critic\_alerts), o lanza un error si no es válida.

*Class LoggerExecute*

* **Ubicación**: src/services
* **Descripción**: Carga y valida datos desde un archivo, y los inserta en la base de datos o caché según su antigüedad.
* **Métodos**:
  + **validate**: Valida el archivo de datos usando DataLoaderFactory.
  + **sync\_old\_logs\_to\_db**: Limpia e inserta logs antiguos desde el caché a la base de datos.
  + **insert\_data**: Separa los logs validados en antiguos (se insertan en base de datos) y recientes (se agregan al caché).

*Class ReportExecutor*

* **Ubicación**: src/services
* **Descripción**: Ejecuta un reporte con una estrategia específica, recopilando datos desde la base o el caché.
* **Métodos**:
  + **get\_log\_data**: Recupera logs desde la base de datos y/o el caché, según el rango de fechas.
  + **execute**: Ejecuta la estrategia de reporte con los datos obtenidos.