Software para lisímetro de pesada para vivero forestal — Memoria técnica (v1.0.0)

1) Introducción y propósito

El software **Lisímetro** es una plataforma en **Python** para monitoreo continuo de peso de sustratos y variables ambientales en viveros forestales, con el fin de **gestionar el riego** de forma trazable y eficiente. Integra:

- **Adquisición de datos** desde celdas de carga y sensores (peso, temperatura, humedad, drenaje, etc.).
- **Almacenamiento estructurado** y seguro de series temporales.
- **Analítica** para cálculo de indicadores, **balance hídrico** y **ETo**.
- **Interfaz web** para visualización, explicación automática y exportación de datos.

2) Objetivo general

Desarrollar un sistema de **lisimetría de pesada de bajo costo** que permita **decidir, validar y optimizar** el riego en vivero forestal con datos de alta frecuencia y trazabilidad completa.

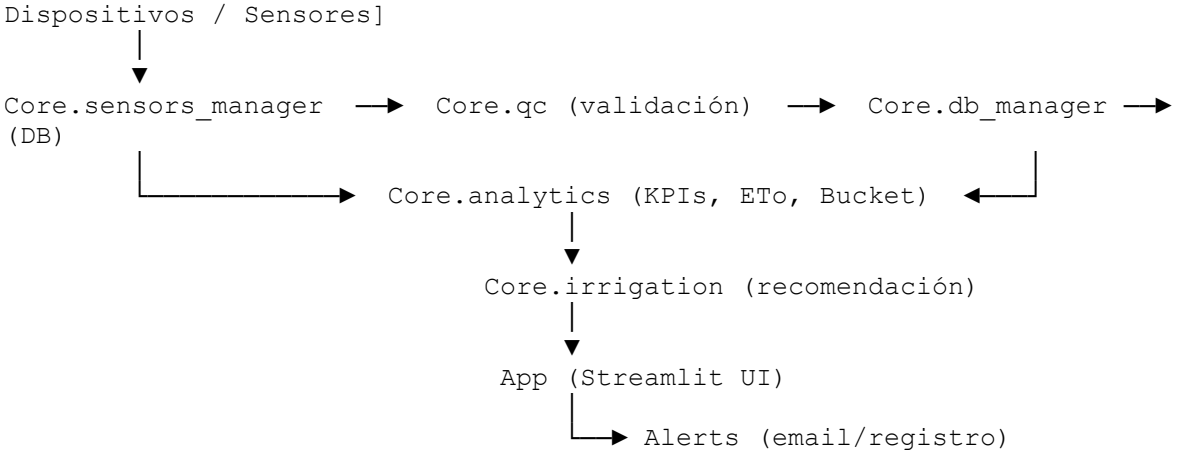
Objetivos específicos

1. Adquirir y registrar en tiempo real **peso**, **eventos de riego**, **drenaje** y **variables climáticas**.
2. Implementar **calibración y tara** reproducibles por maceta/unidad.
3. Calcular **indicadores operativos** (p. ej., CV del peso, pérdidas/ganancias por riego, volumen drenado).
4. Estimar **demanda hídrica** (ETo) y ejecutar un **balance "bucket"** para apoyo a la recomendación.
5. Ofrecer una **UI web** simple para consulta, gráficos, explicaciones automáticas y exportación.
6. Habilitar **alertas** (p. ej., riego anómalo, drenaje excesivo, sensores fuera de rango).

3) Alcance

- **Plataforma**: Python 3.10+; UI con **Streamlit**; almacenamiento **SQLite/PostgreSQL** (SQLAlchemy).
- **Hardware típico**: Raspberry Pi + celdas de carga (HX711 u otro ADC), sensores DHT/DS18B20/ultrasónico (opcional).
- **Interoperabilidad**: exportación CSV/Parquet; importación de series históricas (p. ej., radiación solar).

4) Arquitectura general



5) Estructura del repositorio

```
lisimetro/
├─ src/
│  ├─ app/
│  │  ├─ app.py              # UI principal (Streamlit)
│  │  └─ pages/              # Páginas: Visión general, Datos, Gráficos, Configuración
│  ├─ core/
│  │  ├─ sensors_manager.py     # Lectura de celdas de carga y sensores
│  │  ├─ calibration.py         # Curvas y rutinas de tara/calibración
│  │  ├─ db_manager.py          # Persistencia (SQLAlchemy) y consultas
│  │  ├─ qc.py                  # Control de calidad (filtros, outliers)
│  │  ├─ analytics/
│  │  │  ├─ kpis.py             # Cálculo de indicadores operativos
│  │  │  ├─ eto.py              # ETo (FAO-56; fallback Thornthwaite)
│  │  │  └─ bucket.py           # Balance hídrico tipo “balde”
│  │  ├─ irrigation.py          # Lógica de recomendación
│  │  ├─ alerts.py              # Alertas (SMTP, log)
│  │  └─ scheduler.py           # Tareas periódicas (opcional)
│  └─ utils/
│     ├─ io.py                  # Carga/descarga CSV, Parquet
│     └─ time.py                # Fechas, zonas horarias, agregaciones
├─ data/
│  ├─ raw/                   # CSV crudos (no versionar sin LFS)
│  └─ processed/             # Derivados/limpios
├─ docs/                     # Manuales, protocolos
├─ requirements.txt
└─ README.md
```

6) Descripción de módulos (qué hace cada uno)

6.1 core/sensors_manager.py

Qué hace: abstrae la lectura de dispositivos.

Entradas: configuración de puertos/ADC, intervalos, IDs de macetas.

Salidas: dict normalizado por registro, p. ej.:

```
{
  "ts": "2025-09-14T12:00:05",
  "pot_id": "M01",
  "peso_kg": 7.92,
  "air_temp_c": 23.1,
  "air_rh_pct": 68.0,
  "riego_min": 0.0,
  "drenaje_L": 0.0,
  "source": "sensor"
}
```

Notas: gestiona reconexiones, promedio móvil anti-ruido, debounce de eventos.

6.2 core/calibration.py

Qué hace: rutinas de **tara** y **calibración** (pendiente/offset).

Funciones clave:

- tare(pot_id) → fija peso cero con recipiente vacío.
- fit_curve(pesos_reales, lecturas) → coeficientes a y b (peso = a*lectura + b).
- Guarda histórico de calibraciones por maceta.

6.3 core/qc.py (Control de calidad)

Qué hace: valida y limpia datos.

Reglas típicas:

- Rango físico (peso_kg ≥ 0, 0 ≤ RH ≤ 100).
- Saltos imposibles (Δpeso > umbral).
- Detección de **outliers** (Z-score / MAD).
- Smoothing (media móvil configurable).

6.4 core/db_manager.py

Qué hace: persistencia y consultas.

Tablas mínimas (esquema sugerido):

- measures(ts, pot_id, peso_kg, air_temp_c, air_rh_pct, riego_min, drenaje_L, source)
- pots(pot_id, ubicacion, sustrato, tara_kg, calib_a, calib_b, fc_kg, pwp_kg, area_m2)
- events(ts, pot_id, tipo, valor, meta_json) *(p. ej., "riego_aplicado", "alarma")*

Consultas típicas: last(pot_id), range(pot_id, t0, t1), agregaciones por día/sesión.

6.5 core/analytics/kpis.py

Qué hace: computa indicadores operativos:

- **Peso promedio**, **CV del peso**, **pérdida/ganancia** por ventana (antes/después de riego).
- **Volumen drenado acumulado** y tasa de drenaje.
- **Explicación automática** (texto con interpretaciones simples).

6.6 core/analytics/eto.py

Qué hace: ETo (FAO-56 Penman-Monteith) con fallback **Thornthwaite** si faltan variables.

Entradas: temp máx/mín, HR, viento, radiación; histórico (p. ej., archivo de radiación en data/raw/radiacion.csv).
Salida: ETo diaria/horaria para el **balance hídrico**.

6.7 core/analytics/bucket.py

Qué hace: balance de suelo tipo "balde".

Parámetros: Capacidad de campo (FC) y **Punto de marchitez (PWP)** en **kg** por maceta (o mm si se trabaja por lámina).

Lógica:
S_{t+1} = clamp(S_t + Riego - Drenaje - ETc, PWP, FC) con ETc = Kc * ETo (Kc por especie/etapa).

Salidas: estado hídrico estimado, umbrales y "recomendación base".

6.8 core/irrigation.py

Qué hace: genera una **recomendación de riego** usando:

- estado hídrico del **bucket**;
- tendencia de **peso** reciente;
- reglas operativas (hora/sesión, límites de duración, drenaje máximo).

Salida: duración sugerida (min), notas y motivo.

6.9 core/alerts.py

Qué hace: gestiona **alertas** (registro y, opcionalmente, correo).

Tipos: falta de datos, drenaje excesivo, CV alto anómalo, sensor fuera de rango, etc.

Config: SMTP_HOST, SMTP_PORT, SMTP_USER, SMTP_PASS (cargar desde secrets).

6.10 src/app/app.py (UI)

Pestañas sugeridas:

- **Visión general:** KPIs por maceta/sector, estado de sensores, últimas alertas.
- **Datos:** tabla filtrable, descarga CSV/Parquet.
- **Gráficos:** series de peso, riego, drenaje; comparativas; ventanas pre/post riego.
- **Análisis:** ETo, bucket y **explicación automática** (texto adaptativo).
- **Config:** parámetros de macetas, calibraciones, umbrales, credenciales (no persistentes en repo).

7) Flujo operativo (de punta a punta)

1. **Lectura periódica** de sensores (sensors_manager).
2. **Validación** y **limpieza** (qc).
3. **Almacenamiento** en DB (db_manager).
4. **Cálculo** de KPIs, ETo y **balance** (analytics).
5. **Recomendación** de riego (irrigation) y **alertas** (alerts).
6. **Visualización** y **exportación** desde la **UI**.

8) Indicadores clave (KPIs)

- **Peso promedio** y **CV** en ventana móvil.
- **Ganancia por riego** (Δpeso tras evento).
- **Drenaje total** y **tasa de drenaje**.
- **ETo/ETc** y **estado hídrico** (bucket).
- **Alarmas** por desviaciones (p. ej., CV > umbral).

9) Datos y formatos

- **Mediciones**: CSV/Parquet; timestamps ISO-8601; separador ,.

- **Radiación histórica**: CSV en data/raw/; ruta adaptable (p. ej., /home/lisimetro/lisimetro/data/radiacion.csv).
- **Exportación**: CSV/Parquet con metadatos (pot_id, unidad, fuente).

10) Seguridad y gestión de secretos

- **No** se versionan credenciales.
- Usar /.streamlit/secrets.toml (SMTP/DB) y/o .env.
- **Git LFS** para archivos pesados; evitar subir datos sensibles.

11) Requisitos y despliegue

- **Python 3.10+**
- Paquetes: streamlit, pandas, numpy, matplotlib, sqlalchemy (y driver DB, p. ej. psycopg2-binary).
- **Raspberry Pi** opcional para operación continua.

12) Versionado y calidad

- Convención de commits **Conventional Commits** (feat:, fix:, docs:…).
- Logging con rotación; pruebas básicas de consistencia de datos.

13) Limitaciones conocidas

- La calidad de estimaciones depende de la **calibración** y del **ruido** de sensores.
- Si faltan variables climáticas, ETo recurre a **métodos alternativos** (mayor incertidumbre).
- El **bucket** usa parámetros (FC, PWP, Kc) que requieren ajuste por especie/etapa.