**📚 Documentación de Arquitectura Modular**

**Baseline Adjustment Tool**

**📋 Índice**

1. [Visión General](#visi%C3%B3n-general)
2. [Estructura de Archivos](#estructura-de-archivos)
3. [Los 13 Módulos Explicados](#los-13-m%C3%B3dulos-explicados)
4. [Flujo de Datos](#flujo-de-datos)
5. [Cómo Conectan Entre Ellos](#c%C3%B3mo-conectan-entre-ellos)
6. [Ventajas de Esta Arquitectura](#ventajas-de-esta-arquitectura)
7. [Guía de Mantenimiento](#gu%C3%ADa-de-mantenimiento)
8. [Ejemplos Prácticos](#ejemplos-pr%C3%A1cticos)

**🎯 Visión General**

**Antes: Script Monolítico**

baseline\_adjustment\_original.py (1000+ líneas)

├── Todo mezclado en un solo archivo

├── Difícil de encontrar código específico

├── Cambios en una parte afectan todo

└── Imposible de testear componentes individuales

```

### Ahora: Arquitectura Modular

```

13 módulos organizados por responsabilidad

├── Cada módulo hace UNA cosa

├── Fácil localizar y modificar código

├── Cambios aislados sin efectos colaterales

└── Cada componente es testeable independientemente

```

---

## 📁 Estructura de Archivos

```

baseline\_adjustment\_tool/

│

├── app.py # 🎯 NÚCLEO: Punto de entrada

├── config.py # ⚙️ CONFIGURACIÓN: Constantes

├── session\_manager.py # 💾 ESTADO: Gestión de sesión

│

├── ui/ # 🎨 INTERFAZ DE USUARIO

│ ├── \_\_init\_\_.py

│ ├── sidebar.py # Barra lateral

│ ├── step\_00\_client\_info.py # Paso -1: Info cliente

│ ├── step\_01\_backup.py # Paso 0: Advertencia backup

│ ├── step\_02\_wstd.py # Paso 1: Diagnóstico WSTD

│ ├── step\_03\_kit.py # Paso 2: Standard Kit

│ ├── step\_04\_correction.py # Paso 3: Cálculo corrección

│ ├── step\_05\_baseline.py # Paso 4: Cargar baseline

│ ├── step\_06\_export.py # Paso 5: Exportar

│ └── utilities.py # Utilidades (conversión)

│

├── core/ # ⚡ LÓGICA DE NEGOCIO

│ ├── \_\_init\_\_.py

│ ├── file\_handlers.py # Lectura/escritura archivos

│ ├── spectral\_processing.py # Procesamiento espectral

│ └── report\_generator.py # Generación informes

│

└── utils/ # 🛠️ UTILIDADES

├── \_\_init\_\_.py

├── validators.py # Validaciones

└── plotting.py # Gráficos matplotlib

**🔍 Los 13 Módulos Explicados**

**🎯 NÚCLEO (1 módulo)**

**1. app.py - El Director de Orquesta**

**Responsabilidad:** Coordinar toda la aplicación

python

*# Lo único que hace:*

- Configurar Streamlit

- Inicializar sesión

- Mostrar el paso actual (router)

- Delegar trabajo a otros módulos

**Analogía:** Es como el director de una orquesta. No toca ningún instrumento, solo coordina a todos los músicos.

**Qué NO hace:**

* ❌ No procesa archivos
* ❌ No genera gráficos
* ❌ No hace cálculos
* ❌ No valida datos

**Por qué es útil:**

* ✅ Ves el flujo completo en 50 líneas
* ✅ Agregar un nuevo paso = solo 2 líneas de código
* ✅ Fácil entender qué hace la app

**⚙️ CONFIGURACIÓN (1 módulo)**

**2. config.py - El Diccionario Central**

**Responsabilidad:** Almacenar TODAS las constantes y configuraciones

python

*# Contiene:*

- Textos de mensajes

- Umbrales numéricos

- Configuración de gráficos

- Metadatos por defecto

- Rutas de archivos

**Analogía:** Es como tener todos los ajustes de tu coche en un solo panel de control.

**Ejemplo de uso:**

python

*# En lugar de escribir el texto en cada archivo:*

st.error("No se encontraron mediciones WSTD")

*# Usas:*

st.error(MESSAGES['error\_no\_wstd'])

**Por qué es útil:**

* ✅ Cambiar un mensaje → editas UN solo lugar
* ✅ Traducir la app → solo traduces config.py
* ✅ Ajustar umbrales → no tocas lógica
* ✅ Sin "magic numbers" en el código

**💾 ESTADO (1 módulo)**

**3. session\_manager.py - La Memoria de la App**

**Responsabilidad:** Gestionar el estado de Streamlit

python

*# Maneja:*

- Paso actual del usuario

- Datos del cliente

- Datos de WSTD

- Datos del kit

- Baseline cargado

- Corrección calculada

**Analogía:** Es como la memoria RAM de tu computadora.

**Ejemplo de uso:**

python

*# ANTES (código esparcido por todos lados):*

st.session\_state.client\_data = {'name': 'test'}

if st.session\_state.client\_data is not None:

...

*# AHORA (centralizado y claro):*

save\_client\_data(name='test')

if has\_client\_data():

...

**Por qué es útil:**

* ✅ No accedes directamente a st.session\_state
* ✅ Funciones con nombres descriptivos
* ✅ Validaciones centralizadas
* ✅ Fácil testear

**🎨 INTERFAZ DE USUARIO (8 módulos)**

Cada módulo UI es **una pantalla completa** de la aplicación.

**4. ui/sidebar.py - Barra Lateral**

**Responsabilidad:** Mostrar progreso y navegación

python

*# Muestra:*

- Lista de pasos

- Paso actual resaltado

- Botón "volver atrás"

**5. ui/step\_00\_client\_info.py - Formulario Cliente**

**Responsabilidad:** Recopilar información del cliente

python

*# Hace:*

- Mostrar formulario

- Validar campos obligatorios

- Guardar datos

**6. ui/step\_01\_backup.py - Advertencia Backup**

**Responsabilidad:** Advertir sobre backup de archivos

python

*# Hace:*

- Mostrar advertencia

- Mostrar rutas de archivos

- Confirmar backup

**7. ui/step\_02\_wstd.py - Diagnóstico WSTD**

**Responsabilidad:** Procesar mediciones del White Standard

python

*# Hace:*

- Upload de archivo TSV

- Mostrar espectros

- Calcular métricas diagnósticas

- Visualizar diferencias

**Llama a:**

* file\_handlers.load\_tsv\_file() - Cargar archivo
* spectral\_processing.group\_measurements\_by\_lamp() - Agrupar datos
* plotting.plot\_wstd\_spectra() - Crear gráficos
* validators.validate\_wstd\_measurements() - Validar

**8. ui/step\_03\_kit.py - Standard Kit**

**Responsabilidad:** Procesar mediciones del kit de muestras

python

*# Hace:*

- Upload de archivo TSV

- Selección de lámparas

- Selección de muestras para corrección

- Visualización de espectros

**Llama a:**

* file\_handlers.load\_tsv\_file()
* spectral\_processing.group\_measurements\_by\_lamp()
* spectral\_processing.find\_common\_samples()
* plotting.plot\_kit\_spectra()

**9. ui/step\_04\_correction.py - Cálculo de Corrección**

**Responsabilidad:** Calcular vector de corrección espectral

python

*# Hace:*

- Calcular diferencias espectrales

- Mostrar estadísticas

- Visualizar corrección

- Exportar tabla CSV

**Llama a:**

* spectral\_processing.calculate\_spectral\_correction()
* plotting.plot\_correction\_differences()

**10. ui/step\_05\_baseline.py - Cargar Baseline**

**Responsabilidad:** Cargar archivo baseline (.ref o .csv)

python

*# Hace:*

- Upload de baseline

- Validar dimensiones

- Mostrar metadatos

- Visualizar espectro

**Llama a:**

* file\_handlers.load\_ref\_file() o load\_csv\_baseline()
* validators.validate\_dimension\_match()
* plotting.plot\_baseline\_spectrum()

**11. ui/step\_06\_export.py - Exportar Resultados**

**Responsabilidad:** Aplicar corrección y exportar archivos

python

*# Hace:*

- Aplicar corrección al baseline

- Exportar .ref y .csv corregidos

- Simular efecto de corrección

- Generar informe HTML

**Llama a:**

* spectral\_processing.apply\_baseline\_correction()
* spectral\_processing.simulate\_corrected\_spectra()
* file\_handlers.export\_ref\_file()
* file\_handlers.export\_csv\_file()
* report\_generator.generate\_html\_report()
* plotting.plot\_baseline\_comparison()
* plotting.plot\_corrected\_spectra\_comparison()

**12. ui/utilities.py - Utilidades**

**Responsabilidad:** Conversión de archivos .ref ↔ .csv

python

*# Hace:*

- Convertir .ref a .csv

- Mostrar en expander

**⚡ LÓGICA DE NEGOCIO (3 módulos)**

Estos módulos NO saben nada de Streamlit. Son Python puro.

**13. core/file\_handlers.py - Gestor de Archivos**

**Responsabilidad:** Leer y escribir archivos

python

*# Funciones:*

load\_tsv\_file() *# Leer TSV*

get\_spectral\_columns() *# Extraer columnas #1, #2...*

load\_ref\_file() *# Leer .ref binario*

load\_csv\_baseline() *# Leer .csv baseline*

export\_ref\_file() *# Escribir .ref*

export\_csv\_file() *# Escribir .csv*

**Por qué es útil:**

* ✅ Toda la lógica de archivos en UN lugar
* ✅ Fácil cambiar formato de archivo
* ✅ Reutilizable en otros proyectos
* ✅ Testeable sin Streamlit

**14. core/spectral\_processing.py - Procesador Espectral**

**Responsabilidad:** Operaciones sobre datos espectrales

python

*# Funciones:*

group\_measurements\_by\_lamp() *# Agrupar por lámpara*

find\_common\_samples() *# Encontrar muestras comunes*

calculate\_spectral\_correction() *# Calcular corrección*

apply\_baseline\_correction() *# Aplicar corrección*

simulate\_corrected\_spectra() *# Simular resultado*

**Por qué es útil:**

* ✅ Toda la lógica científica en UN lugar
* ✅ Fácil modificar algoritmos
* ✅ Testeable con datos sintéticos
* ✅ Sin dependencia de UI

**15. core/report\_generator.py - Generador de Informes**

**Responsabilidad:** Crear informe HTML

python

*# Funciones:*

generate\_html\_report() *# Generar informe completo*

generate\_wstd\_section() *# Sección WSTD*

generate\_process\_details() *# Detalles proceso*

generate\_samples\_table() *# Tabla muestras*

generate\_correction\_statistics() *# Estadísticas*

generate\_charts\_section() *# Gráficos embebidos*

**Por qué es útil:**

* ✅ HTML separado de la lógica
* ✅ Fácil cambiar diseño del informe
* ✅ Posible generar PDF/Word en futuro
* ✅ Reutilizable para otros informes

**🛠️ UTILIDADES (2 módulos)**

Funciones auxiliares usadas por varios módulos.

**16. utils/validators.py - Validador**

**Responsabilidad:** Validar datos

python

*# Funciones:*

validate\_wstd\_measurements() *# Validar WSTD*

validate\_common\_samples() *# Validar muestras comunes*

validate\_dimension\_match() *# Validar dimensiones*

**Por qué es útil:**

* ✅ Validaciones consistentes
* ✅ Fácil agregar nuevas validaciones
* ✅ Mensajes de error centralizados

**17. utils/plotting.py - Graficador**

**Responsabilidad:** Crear gráficos matplotlib

python

*# Funciones:*

plot\_wstd\_spectra() *# Gráfico WSTD*

plot\_correction\_differences() *# Diferencias espectrales*

plot\_baseline\_comparison() *# Baseline original vs corregido*

plot\_corrected\_spectra\_comparison() *# Resultado corrección*

plot\_kit\_spectra() *# Espectros del kit*

```

\*\*Por qué es útil:\*\*

- ✅ Estilo consistente de gráficos

- ✅ Configuración centralizada (tamaños, colores)

- ✅ Fácil cambiar diseño de gráficos

- ✅ Posible cambiar a plotly/bokeh sin tocar UI

---

*## 🔄 Flujo de Datos*

*### Ejemplo: Cargar archivo WSTD*

```

Usuario → [UI] → [Core] → [Utils] → [Session]

↓ ↓ ↓ ↓ ↓

Click step\_02 file\_ plotting save\_wstd

upload \_wstd handlers \_data()

.py .py

```

\*\*Paso a paso:\*\*

1. \*\*Usuario\*\* sube archivo en `step\_02\_wstd.py`

2. \*\*UI\*\* llama a `file\_handlers.load\_tsv\_file(file)`

3. \*\*Core\*\* procesa el archivo y devuelve DataFrame

4. \*\*UI\*\* llama a `spectral\_processing.group\_measurements\_by\_lamp()`

5. \*\*Core\*\* agrupa datos y devuelve resultado

6. \*\*UI\*\* llama a `validators.validate\_wstd\_measurements()`

7. \*\*Utils\*\* valida y devuelve True/False

8. \*\*UI\*\* llama a `plotting.plot\_wstd\_spectra()`

9. \*\*Utils\*\* genera gráfico

10. \*\*UI\*\* muestra gráfico con `st.pyplot()`

11. \*\*UI\*\* llama a `session\_manager.save\_wstd\_data()`

12. \*\*Session\*\* guarda en `st.session\_state`

---

*## 🔗 Cómo Conectan Entre Ellos*

*### Diagrama de Dependencias*

```

app.py

|

+-------------+-------------+

| | |

sidebar.py step\_XX.py utilities.py

|

+-------------+-------------+

| | |

file\_handlers spectral\_ report\_

.py processing.py generator.py

|

+-------------+-------------+

| |

validators.py plotting.py

```

*### Reglas de Conexión*

1. \*\*app.py\*\* → Solo importa módulos UI

2. \*\*Módulos UI\*\* → Pueden importar Core, Utils, Config, Session

3. \*\*Módulos Core\*\* → Pueden importar Utils, Config (NO UI, NO Session)

4. \*\*Módulos Utils\*\* → Pueden importar Config (NO UI, NO Core, NO Session)

5. \*\*Config\*\* → NO importa nada (solo constantes)

6. \*\*Session\*\* → NO importa nada excepto Streamlit

*### Flujo Unidireccional*

```

UI → Core → Utils

↓ ↓ ↓

Session ← ← ←

**Nunca:**

* Core no llama a UI
* Utils no llama a Core ni UI
* Config no llama a nadie

**✨ Ventajas de Esta Arquitectura**

**1. Mantenibilidad**

**Escenario: Cambiar un mensaje de error**

**ANTES (monolítico):**

python

*# Buscar en 1000 líneas...*

*# Línea 234: st.error("❌ No se encontraron mediciones WSTD")*

*# Línea 567: st.error("❌ No se encontraron mediciones WSTD")*

*# Línea 789: st.error("❌ No se encontraron mediciones WSTD")*

*# Cambiar 3 lugares diferentes*

**AHORA (modular):**

python

*# Abrir config.py*

*# Cambiar 1 línea:*

'error\_no\_wstd': "❌ No hay datos WSTD en el archivo"

*# Listo. Se actualiza en toda la app.*

**2. Escalabilidad**

**Escenario: Agregar nuevo paso**

**ANTES (monolítico):**

python

*# Insertar 200 líneas en medio del archivo*

*# Ajustar todos los if/elif*

*# Riesgo de romper otros pasos*

**AHORA (modular):**

python

*# 1. Crear ui/step\_07\_nuevo.py (archivo nuevo)*

*# 2. En app.py agregar:*

elif current\_step == 6:

render\_nuevo\_step()

*# 3. En config.py agregar nombre del paso*

*# Listo. No tocaste código existente.*

**3. Testabilidad**

**Escenario: Probar cálculo de corrección**

**ANTES (monolítico):**

python

*# Imposible testear sin ejecutar toda la app*

*# No puedes probar solo el cálculo*

**AHORA (modular):**

python

*# test\_spectral\_processing.py*

def test\_calculate\_correction():

df\_ref = create\_test\_data()

df\_new = create\_test\_data()

correction = calculate\_spectral\_correction(df\_ref, df\_new)

assert len(correction) == 256

assert not np.isnan(correction).any()

**4. Reutilización**

**Escenario: Usar el procesamiento en otra app**

**ANTES (monolítico):**

python

*# Copiar y pegar 1000 líneas*

*# Quitar todo lo de Streamlit manualmente*

*# Rezar para que funcione*

**AHORA (modular):**

python

*# En tu nueva app:*

from baseline\_tool.core.spectral\_processing import calculate\_spectral\_correction

*# Listo. Es Python puro, funciona en cualquier lado.*

**5. Colaboración**

**Escenario: Dos personas trabajando**

**ANTES (monolítico):**

python

*# Persona A edita línea 300*

*# Persona B edita línea 400*

*# Git merge conflict 💥*

**AHORA (modular):**

python

*# Persona A trabaja en ui/step\_02\_wstd.py*

*# Persona B trabaja en core/report\_generator.py*

*# No hay conflictos, archivos diferentes*

**6. Debugging**

**Escenario: Error en carga de archivo**

**ANTES (monolítico):**

python

*# Error en línea 456*

*# ¿Qué función es?*

*# ¿Dónde empieza?*

*# ¿Qué variables usa?*

*# Buscar en 1000 líneas...*

**AHORA (modular):**

python

*# Error en file\_handlers.py, línea 23*

*# Abres el archivo, son solo 150 líneas*

*# Función load\_tsv\_file() claramente identificada*

*# Variables locales, fácil de depurar*

```

---

*## 📚 Guía de Mantenimiento*

*### Modificaciones Comunes*

*#### 1. Cambiar un texto o mensaje*

```

Ubicación: config.py

Sección: INSTRUCTIONS o MESSAGES

Impacto: Toda la app

Dificultad: ⭐ Muy fácil

```

*#### 2. Cambiar un umbral numérico*

```

Ubicación: config.py

Sección: WSTD\_THRESHOLDS o similares

Impacto: Solo el módulo que lo usa

Dificultad: ⭐ Muy fácil

```

*#### 3. Modificar un gráfico*

```

Ubicación: utils/plotting.py

Función específica: plot\_xxx()

Impacto: Solo donde se use ese gráfico

Dificultad: ⭐⭐ Fácil

```

*#### 4. Cambiar algoritmo de cálculo*

```

Ubicación: core/spectral\_processing.py

Función específica: calculate\_xxx()

Impacto: Solo ese cálculo

Dificultad: ⭐⭐⭐ Medio

```

*#### 5. Agregar nueva validación*

```

Ubicación: utils/validators.py

Crear nueva función: validate\_xxx()

Llamarla desde: ui/step\_XX.py

Dificultad: ⭐⭐ Fácil

```

*#### 6. Agregar nuevo paso*

```

1. Crear: ui/step\_XX\_nombre.py

2. Modificar: app.py (agregar elif)

3. Modificar: config.py (agregar nombre paso)

Dificultad: ⭐⭐⭐ Medio

```

*#### 7. Cambiar formato de archivo*

```

Ubicación: core/file\_handlers.py

Modificar: load\_xxx() o export\_xxx()

Impacto: Todos los pasos que usen ese formato

Dificultad: ⭐⭐⭐⭐ Difícil

**Reglas de Oro**

1. **Un archivo, una responsabilidad**
   * Si un archivo hace más de una cosa, divídelo
2. **Funciones pequeñas**
   * Máximo 50 líneas por función
   * Si es más larga, divídela
3. **Sin lógica en UI**
   * UI solo muestra y recibe input
   * Toda lógica va en Core o Utils
4. **Importaciones claras**
   * Solo importa lo que necesitas
   * Sigue las reglas de dependencias
5. **Nombres descriptivos**
   * calculate\_spectral\_correction() ✅
   * process\_data() ❌

**💡 Ejemplos Prácticos**

**Ejemplo 1: Agregar nueva métrica en diagnóstico WSTD**

**Requisito:** Mostrar la mediana además de la media

python

*# 1. En ui/step\_02\_wstd.py, función render\_diagnostic\_metrics()*

*# AGREGAR después de la línea de mean\_val:*

median\_val = np.median(np.abs(spectrum))

*# AGREGAR después del metric de "Desv. media":*

st.metric("Mediana", f"{median\_val:.4f}")

**Archivos modificados:** 1  
**Líneas añadidas:** 2  
**Tiempo:** 2 minutos

**Ejemplo 2: Cambiar el umbral de "bien ajustado"**

**Requisito:** Cambiar de 0.01 a 0.005

python

*# En config.py, línea ~60*

WSTD\_THRESHOLDS = {

'good': 0.005, *# Cambiar de 0.01 a 0.005*

'warning': 0.05,

'bad': float('inf')

}

**Archivos modificados:** 1  
**Líneas modificadas:** 1  
**Tiempo:** 30 segundos

**Ejemplo 3: Agregar nuevo formato de exportación (PDF)**

python

*# 1. Crear core/pdf\_exporter.py*

def export\_pdf\_report(data):

*# Lógica para generar PDF*

return pdf\_bytes

*# 2. En ui/step\_06\_export.py, agregar botón:*

if st.button("📥 Descargar informe PDF"):

from core.pdf\_exporter import export\_pdf\_report

pdf = export\_pdf\_report(kit\_data)

st.download\_button("Download", pdf, "informe.pdf")

**Archivos creados:** 1  
**Archivos modificados:** 1  
**Impacto:** Solo en exportación

**Ejemplo 4: Cambiar biblioteca de gráficos (matplotlib → plotly)**

python

*# Solo modificar utils/plotting.py*

*# Cambiar todas las funciones plot\_xxx()*

*# ANTES:*

import matplotlib.pyplot as plt

def plot\_wstd\_spectra(...):

fig, ax = plt.subplots()

ax.plot(...)

return fig

*# DESPUÉS:*

import plotly.graph\_objects as go

def plot\_wstd\_spectra(...):

fig = go.Figure()

fig.add\_trace(go.Scatter(...))

return fig

**Archivos modificados:** 1 (utils/plotting.py)  
**UI no se toca:** Solo cambia st.pyplot() → st.plotly\_chart()  
**Lógica no se toca:** Core y Utils siguen igual

**🎓 Resumen Final**

**Lo que logramos:**

| **Aspecto** | **Antes** | **Ahora** |
| --- | --- | --- |
| **Archivos** | 1 archivo (1000+ líneas) | 13 módulos organizados |
| **Encontrar código** | Buscar en todo el archivo | Saber exactamente dónde está |
| **Hacer cambios** | Riesgo alto de romper algo | Cambios aislados y seguros |
| **Testing** | Imposible sin la app completa | Cada módulo testeable |
| **Colaboración** | Conflictos constantes | Trabajo paralelo sin problemas |
| **Reutilización** | Copiar/pegar masivo | Importar módulos específicos |
| **Mantenimiento** | Difícil y lento | Rápido y seguro |

**Los 13 Módulos en Una Frase:**

1. **app.py** - Dirige la orquesta
2. **config.py** - Almacén de constantes
3. **session\_manager.py** - Memoria de la app
4. **sidebar.py** - Muestra progreso
5. **step\_00\_client\_info.py** - Formulario cliente
6. **step\_01\_backup.py** - Advertencia backup
7. **step\_02\_wstd.py** - Diagnóstico WSTD
8. **step\_03\_kit.py** - Procesa kit de muestras
9. **step\_04\_correction.py** - Calcula corrección
10. **step\_05\_baseline.py** - Carga baseline
11. **step\_06\_export.py** - Exporta resultados
12. **utilities.py** - Convierte archivos
13. **file\_handlers.py** - Lee/escribe archivos
14. **spectral\_processing.py** - Procesa espectros
15. **report\_generator.py** - Genera informes
16. **validators.py** - Valida datos
17. **plotting.py** - Crea gráficos

**Principio Fundamental:**

**"Cada módulo hace UNA cosa, y la hace bien"**

* Unix Philosophy

Esta arquitectura no es complicada, es **organizada**.  
No es sobre tener más archivos, es sobre tener **responsabilidades claras**.