Reporte de Inconsistencias de Diseño

Dashboard Pletorica - Análisis de Código

Fecha: 2026-01-13 | Analizado por: Claude Code

Resumen Ejecutivo

Se encontraron 15 categorías de inconsistencias en el código del proyecto.

Severidad	Cantidad	Descripción
CRÍTICA	3	Afectan funcionamiento
MEDIA	4	Inconsistencias de patrones
MENOR	8	Diferencias estéticas

INCONSISTENCIAS CRÍTICAS

1. Event Handlers: async vs sync

Problema: Los manejadores de eventos usan patrones completamente diferentes entre módulos.

EMPRESAS (empresas_state.py:516-519):

```
async def handle_key_down(self, key):
    if key == "Enter":
        await self.aplicar_filtros()
```

TIPO SERVICIO (tipo_servicio_state.py:153-156):

```
def handle_key_down(self, key: str):  # NO async
    if key == "Enter":
        return TipoServicioState.buscar_tipos
```

Recomendación: Estandarizar todos como async/await.

2. Manejo de Errores: Centralizado vs Inline

Problema: Empresas tiene helper centralizado, Tipo Servicio repite código.

EMPRESAS tiene _manejar_error() centralizado:

```
def _manejar_error(self, error, operacion=""):
    if isinstance(error, DuplicateError):
        self.mensaje_info = f"El campo ya existe"
    # ... manejo centralizado
```

TIPO SERVICIO repite en cada método:

```
except DuplicateError as e:
    self.error_clave = f"La clave ya existe"
except DatabaseError as e:
    self.mostrar_mensaje(...)
```

Recomendación: Implementar _manejar_error() en BaseState.

3. Variables de Estado Inconsistentes

Módulo	Variable	Archivo
Empresas	saving	empresas_state.py:53
Tipo Servicio	loading, saving	tipo_servicio_state.py:32
Simulador	is_calculating	simulador_state.py:38

Recomendación: Estandarizar: loading, saving, processing

INCONSISTENCIAS MEDIAS

4. Patrón CRUD: Directo vs Wrapper

EMPRESAS - Métodos directos:

```
async def crear_empresa(self):
async def actualizar_empresa(self):
```

TIPO SERVICIO - Wrapper con privados:

```
async def guardar_tipo(self):     # wrapper
async def _crear_tipo(self):     # privado
async def _actualizar_tipo(self): # privado
```

Recomendación: Adoptar patrón wrapper en todos los módulos.

5. Modelo Resumen Faltante

EmpresaResumen existe pero TipoServicioResumen NO existe.

Tipo Servicio usa conversión manual: tipo.model_dump()

Recomendación: Crear TipoServicioResumen.

6. Normalización en Diferentes Momentos

EMPRESAS: Normaliza en _normalizar_texto() al guardar

TIPO SERVICIO: Normaliza en setters (inmediato)

Recomendación: Normalizar en setters para feedback inmediato.

7. Estado de Modal: String vs Boolean

EMPRESAS: modo_modal_empresa: str = '' ("crear"/"editar")

TIPO SERVICIO: es_edicion: bool = False

Recomendación: Usar string para extensibilidad.

INCONSISTENCIAS MENORES

8. Imports

Relative vs Absolute imports

9. Métodos limpieza

limpiar_formulario() vs _limpiar_formulario()

10. @rx.var

puede_guardar falta en Empresas

11. Enums

EstatusEmpresa tiene SUSPENDIDO, TipoServicio no

12. Dataclasses

costo_patronal usa dataclass, resto Pydantic

13. Servicios

TipoServicio tiene más métodos helper

14. BaseState

loading redefinido en cada State

15. Validators

Funciones batch con firmas diferentes

Plan de Acción Recomendado

Fase 1 - Críticas (Inmediato)

[ ] Estandarizar event handlers a async/await

[ ] Implementar _manejar_error() en BaseState

[ ] Renombrar variables de estado de carga

Fase 2 - Medias (Corto plazo)

[ ] Adoptar patrón wrapper CRUD en Empresas

[ ] Crear TipoServicioResumen

[ ] Mover normalización a setters en Empresas

Fase 3 - Menores (Cuando sea conveniente)

[ ] Unificar estilo de imports

[ ] Renombrar métodos de limpieza

[ ] Migrar costo_patronal a Pydantic