Reporte de Codigo Redundante

Dashboard Pletorica - Analisis de Codigo

RESUMEN EJECUTIVO

Se identificaron aproximadamente 386 lineas de codigo redundante o duplicado en el proyecto. Este reporte detalla cada categoria con ejemplos y recomendaciones.

Estadisticas:

- Lineas duplicadas identificadas: ~386
- Archivos afectados: 8+
- Categorias de duplicacion: 6

1. MANEJO DE ERRORES REPETIDO (~80 lineas)

Problema:

El bloque try/except se repite en cada metodo CRUD de TipoServicioState, mientras que EmpresasState tiene un metodo centralizado _manejar_error().

Archivos afectados:

- tipo_servicio_state.py: lineas 218-227, 285-295, 309-312

Codigo duplicado (repetido 4 veces):

```
except DuplicateError as e:
    self.error_clave = f"La clave ya existe"
except NotFoundError as e:
    self.mostrar_mensaje(str(e), "error")
except DatabaseError as e:
    self.mostrar_mensaje(f"Error de BD: {str(e)}", "error")
except Exception as e:
    self.mostrar_mensaje(f"Error: {str(e)}", "error")
```

Solucion recomendada:

Implementar _manejar_error() en TipoServicioState o mejor, en BaseState para uso compartido.

Reporte de Codigo Redundante

Dashboard Pletorica - Analisis de Codigo

2. SETTERS EXPLICITOS REPETIDOS (~60 lineas)

Problema:

Cada State tiene setters explicitos casi identicos para campos de formulario.

Archivos afectados:

- empresas_state.py: lineas 83-134 (15 setters)
- tipo_servicio_state.py: lineas 57-78 (8 setters)

Patron repetido:

```
def set_form_nombre(self, value: str):
    self.form_nombre = value

def set_form_clave(self, value: str):
    self.form_clave = value.upper() if value else ""
```

Solucion recomendada:

Usar un decorador o metaclase que genere setters automaticamente, o un helper generico en BaseState.

3. VALIDACION DE CAMPOS REPETIDA (~50 lineas)

Problema:

Metodos de validacion en tiempo real son casi identicos entre modulos.

Archivos afectados:

- empresas_state.py: lineas 332-364 (6 validadores)
- tipo_servicio_state.py: lineas 83-99 (3 validadores)

Patron repetido:

```
def validar_nombre_campo(self):
    self.error_nombre = validar_nombre(self.form_nombre)

def validar_clave_campo(self):
    self.error_clave = validar_clave(self.form_clave)
```

Solucion recomendada:

Crear un mixin o helper generico para validacion en tiempo real.

Reporte de Codigo Redundante

Dashboard Pletorica - Analisis de Codigo

4. OPERACIONES DE MODAL DUPLICADAS (~40 lineas)

Problema:

abrir_modal_crear, cerrar_modal, limpiar_formulario siguen patrones casi identicos.

Archivos afectados:

- empresas_state.py: lineas 256-298
- tipo_servicio_state.py: lineas 166-198

Patron repetido:

```
def abrir_modal_crear(self):
    self.limpiar_formulario()
    self.modo_modal = "crear"
    self.mostrar_modal = True

def cerrar_modal(self):
    self.mostrar_modal = False
    self.limpiar_formulario()
```

Solucion recomendada:

Crear ModalMixin en BaseState con logica generica de modales.

5. CARGA DE DATOS CON TRY/EXCEPT (~30 lineas)

Problema:

cargar_empresas y cargar_tipos tienen estructura identica.

Patron repetido:

```
async def cargar_datos(self):
    self.loading = True
    try:
        self.datos = await servicio.obtener_todas(...)
    except DatabaseError as e:
        self.mostrar_mensaje(f"Error: {e}", "error")
        self.datos = []
    finally:
        self.loading = False
```

Solucion recomendada:

Crear decorador @with_loading que maneje el patron loading/try/except/finally.

6. PROPIEDADES CALCULADAS SIMILARES (~26 lineas)

Problema:

tiene_errores_formulario es casi identico entre modulos.

Patron repetido:

```
@rx.var
def tiene_errores_formulario(self) -> bool:
    return bool(
        self.error_campo1 or
        self.error_campo2 or
        self.error_campo3
    )
```

Solucion recomendada:

Definir lista de campos de error y verificar dinamicamente en BaseState.

Reporte de Codigo Redundante

Dashboard Pletorica - Analisis de Codigo

PLAN DE ACCION RECOMENDADO

Fase 1 - Alta prioridad (reduce ~130 lineas):

- [ ] Implementar _manejar_error() en BaseState
- [ ] Crear ModalMixin con logica generica
- [ ] Crear decorador @with_loading

Fase 2 - Media prioridad (reduce ~110 lineas):

- [ ] Crear ValidacionMixin generico
- [ ] Crear helper para setters con transformacion

Fase 3 - Mejoras opcionales (reduce ~46 lineas):

- [ ] Refactorizar propiedades calculadas
- [ ] Unificar estructura de filtros

ARCHIVOS AFECTADOS

empresas_state.py	#1, #2, #3, #4, #5, #6
tipo_servicio_state.py	#1, #2, #3, #4, #5, #6
base_state.py	Destino de refactorizacion
empresas_validators.py	#3
tipo_servicio_validators.py	#3

Generado por Claude Code - Dashboard Pletorica