Nueva Arquitectura

Dashboard Pletorica

Simplificacion Arquitectural - Acceso Directo a Supabase

Enero 2026

- 63 archivos modificados
- ~3,200 lineas netas eliminadas
- 7 repositorios eliminados, 4 mantenidos
- 9 metodos de servicio sin uso removidos
- 14 componentes UI sin uso removidos

1. Resumen del Cambio

Antes (3 capas para todos los modulos)

```
Presentation  -->  Services  -->  Repositories  -->  Supabase
                      |              |
                  Entities       Entities

11 servicios, 11 repositorios (todos los modulos)
```

Despues (Hibrido: 2 o 3 capas segun complejidad)

```
Modulos SIMPLES (7):
  Presentation  -->  Services  -->  Supabase (directo)

Modulos COMPLEJOS (4):
  Presentation  -->  Services  -->  Repositories  -->  Supabase

11 servicios, 4 repositorios (solo los complejos)
```

El criterio para mantener un repositorio es la complejidad de las queries: JOINs manuales, agregaciones multi-tabla, filtros de 8+ parametros, o creates que involucran N+M inserts. Los modulos simples (CRUD basico) ahora acceden a Supabase directamente desde el servicio.

2. Nueva Estructura de Archivos

```
app/
|-- core/                          # Sin cambios
|-- entities/                      # Sin cambios (4 clases sin uso eliminadas)
|-- database/                      # Sin cambios
|
|-- repositories/                  # REDUCIDO: solo 4 repos complejos
|   |-- __init__.py
|   |-- plaza_repository.py        # 700+ lineas, JOINs, agregaciones
|   |-- requisicion_repository.py  # Items/partidas, creates N+M
|   |-- empleado_repository.py     # Multi-tabla, filtros complejos
|   |-- contrato_repository.py     # buscar_con_filtros 8+ params
|
|-- services/                      # 7 servicios con Supabase directo
|   |-- empresa_service.py         # Supabase directo
|   |-- tipo_servicio_service.py   # Supabase directo
|   |-- categoria_puesto_service.py # Supabase directo
|   |-- archivo_service.py         # Supabase directo
|   |-- pago_service.py            # Supabase directo
|   |-- contrato_categoria_service.py  # Supabase directo
|   |-- historial_laboral_service.py   # Supabase directo
|   |
|   |-- empleado_service.py        # Usa repositorio
|   |-- contrato_service.py        # Usa repositorio
|   |-- plaza_service.py           # Usa repositorio
|   |-- requisicion_service.py     # Usa repositorio
|
|-- presentation/                  # Imports limpiados
|-- app.py
```

3. Patron de Servicio con Supabase Directo

Antes: Servicio delegaba al repositorio

```
class EmpresaService:
    def __init__(self):
        self.repository = SupabaseEmpresaRepository()

    async def obtener_por_id(self, empresa_id):
        return await self.repository.obtener_por_id(empresa_id)

class SupabaseEmpresaRepository:
    def __init__(self):
        self.supabase = db_manager.get_client()
        self.tabla = "empresas"

    async def obtener_por_id(self, empresa_id):
        result = self.supabase.table(self.tabla)\n            .select("*").eq("id", empresa_id).execute
        if not result.data:
            raise NotFoundError(...)
        return Empresa(**result.data[0])
```

Despues: Servicio accede a Supabase directamente

```
class EmpresaService:
    def __init__(self):
        self.supabase = db_manager.get_client()
        self.tabla = "empresas"

    async def obtener_por_id(self, empresa_id):
        try:
            result = self.supabase.table(self.tabla)\n                .select("*").eq("id", empresa_id)
            if not result.data:
                raise NotFoundError(...)
            return Empresa(**result.data[0])
        except NotFoundError:
            raise
        except Exception as e:
            raise DatabaseError(...)
```

El servicio ahora es responsable tanto de la logica de negocio como del acceso a datos. El manejo de excepciones (NotFoundError, DuplicateError, DatabaseError) se preserva intacto.

4. Repositorios Eliminados vs Mantenidos

7 Repositorios Eliminados

Estos repositorios solo hacian proxy de queries CRUD simples:

Repositorio Eliminado	Absorbido por	Tabla
empresa_repository	empresa_service	empresas
tipo_servicio_repository	tipo_servicio_service	tipos_servicio
categoria_puesto_repo	categoria_puesto_svc	categorias_puesto
archivo_repository	archivo_service	archivo_sistema
pago_repository	pago_service	pagos
contrato_categoria_repo	contrato_categ_service	contrato_categorias
historial_laboral_repo	historial_laboral_svc	historial_laboral

4 Repositorios Mantenidos

Estos repositorios tienen queries complejas que justifican la separacion:

Repositorio	Justificacion
plaza_repository	700+ lineas, JOINs manuales, agregaciones
requisicion_repository	Creates con items/partidas (N+M inserts)
empleado_repository	Multi-tabla, filtros complejos
contrato_repository	buscar_con_filtros con 8+ parametros

5. Codigo Muerto Eliminado

Metodos de Servicio sin Callers (9)

empresa_service: obtener_resumen_empresas

contrato_service: marcar_vencido, validar_montos, validar_fechas

empleado_service: crear_con_clave_auto, cambiar_empresa

categoria_puesto_service: cambiar_orden

pago_service: buscar_por_rango_fechas, obtener_resumen_lista

Archivos UI Eliminados (4)

- empresa_card.py - Sistema de cards nunca integrado en la pagina
- empresa_grid.py - Grid de empresas nunca usado
- barra_herramientas.py - Toolbar nunca llamado desde paginas
- buttons.py - boton_accion/acciones_crud sin callers

Funciones UI Eliminadas (10)

- view_toggle_with_label(), view_toggle_segmented()
- status_badge_contrato(), status_badge_entidad(), status_badge_plaza(), status_dot()
- breadcrumb(), breadcrumb_item()

Otros

- 4 entidades Pydantic sin uso (ArchivoSistemaCreate, ArchivoSistemaResumen, etc.)
- Alias deprecado MotivoFinHistorial = TipoMovimiento
- ~60 imports sin usar en 31 archivos
- 3 archivos huerfanos + 3 directorios de tests vacios

6. Flujo de Dependencias

Modulos Simples (7 servicios)

```
Presentation (State)
    |   from app.services import empresa_service
    v
Service (empresa_service.py)
    |   self.supabase = db_manager.get_client()
    |   self.supabase.table("empresas").select(...)
    v
Supabase (PostgREST API)
    |
    v
PostgreSQL
```

Modulos Complejos (4 servicios)

```
Presentation (State)
    |   from app.services import plaza_service
    v
Service (plaza_service.py)
    |   self.repository = SupabasePlazaRepository()
    v
Repository (plaza_repository.py)
    |   JOINs, agregaciones, filtros multi-campo
    v
Supabase (PostgREST API)
    |
    v
PostgreSQL
```

Reglas de Dependencia

- Presentation SOLO importa de Services y Entities
- Services simples importan de Database y Entities directamente
- Services complejos importan de Repositories y Entities
- Repositories importan de Database y Entities
- Entities son puras (sin dependencias)

7. Metricas del Refactoring

Metrica	Antes	Despues
Archivos de repositorio	11	4
Archivos de servicio	11	11
Lineas totales (approx)	+6,700	+3,500
Componentes UI	~25 funciones	~15 funciones
Imports sin usar	~60	0
Metodos sin callers	~20	0

Beneficios

- Menos indirecciones: Los modulos simples van directo a Supabase sin proxy
- Menos archivos: 17 archivos eliminados, menos superficie de mantenimiento
- Codigo mas limpio: Sin imports muertos, sin metodos sin callers
- Misma funcionalidad: Cero regresiones, la app compila y arranca correctamente
- Escalable: Los modulos complejos mantienen su repositorio dedicado

Verificacion

- 11 servicios importan correctamente
- 4 repositorios importan correctamente
- AST check en 15 archivos modificados: sin errores de sintaxis
- Cero referencias a repositorios eliminados en codigo
- Reflex compila 61/61 paginas y la app arranca sin errores