Modulo Historial Laboral

Dashboard Pletorica - Generado: 2026-01-21 16:30

1. Descripcion General

El modulo de Historial Laboral es una bitacora automatica que registra todos los movimientos de empleados en el sistema. Este modulo es de SOLO LECTURA - los registros se crean automaticamente cuando ocurren eventos relevantes en el modulo de empleados.

Caracteristicas principales:

- Registro automatico de movimientos (alta, baja, suspension, etc.)
- Solo lectura - no hay formularios de creacion/edicion
- Historial completo de cada empleado
- Seguimiento de asignaciones a plazas

2. Arquitectura del Modulo

El modulo sigue la arquitectura de capas del proyecto:

```
app/
  entities/historial_laboral.py        # Entidades de dominio
  repositories/historial_laboral_repository.py  # Acceso a datos
  services/historial_laboral_service.py         # Logica de negocio
  presentation/pages/historial_laboral/         # UI (Reflex)
    historial_laboral_state.py
    historial_laboral_page.py
    historial_laboral_modals.py
```

3. Enums (app/core/enums.py)

EstatusHistorial

Representa el estado del empleado en el registro de historial:

- ACTIVO - Empleado activo con plaza asignada
- INACTIVO - Empleado sin plaza o dado de baja
- SUSPENDIDO - Empleado suspendido temporalmente

TipoMovimiento

Tipos de movimiento que se registran automaticamente:

- ALTA - Cuando se crea un nuevo empleado

- ASIGNACION - Cuando se asigna empleado a una plaza
- CAMBIO_PLAZA - Cuando cambia de una plaza a otra
- SUSPENSION - Cuando se suspende un empleado
- REACTIVACION - Cuando se reactiva un empleado suspendido
- BAJA - Cuando se da de baja un empleado

4. Entidades (app/entities/historial_laboral.py)

HistorialLaboral

Modelo principal que representa un registro de historial:

```
class HistorialLaboral(BaseModel):
    id: int
    empleado_id: int
    plaza_id: Optional[int]  # NULL si no tiene plaza
    tipo_movimiento: Optional[TipoMovimiento]
    fecha_inicio: date
    fecha_fin: Optional[date]  # NULL si es registro activo
    estatus: EstatusHistorial
    notas: Optional[str]
```

HistorialLaboralInterno

Modelo interno usado solo por el servicio para crear registros. NO se expone en la UI.

HistorialLaboralResumen

Modelo enriquecido para listados en UI. Incluye datos del empleado, plaza, categoria, contrato y empresa.

5. Repositorio (app/repositories/historial_laboral_repository.py)

Metodos de Lectura (para UI)

- obtener_por_id(historial_id) - Obtiene un registro por ID
- obtener_por_empleado(empleado_id) - Historial completo de un empleado
- obtener_todos(filtros) - Lista con filtros opcionales
- obtener_registro_activo(empleado_id) - Registro actual sin fecha_fin
- contar(filtros) - Cuenta registros con filtros

Metodos de Escritura (internos)

- crear(datos) - Crea nuevo registro (solo desde servicio)
- cerrar_registro(id, fecha_fin) - Cierra un registro
- cerrar_registro_activo(empleado_id, fecha_fin) - Cierra registro activo

6. Servicio (app/services/historial_laboral_service.py)

Metodos de Lectura (para UI)

Estos metodos se usan en la pagina de historial laboral:

- obtener_por_id(historial_id)
- obtener_por_empleado(empleado_id, limite)
- obtener_todos(empleado_id, estatus, limite, offset)
- contar(empleado_id, estatus)
- obtener_registro_activo(empleado_id)

Metodos Automaticos (llamados desde empleado_service)

Estos metodos se llaman automaticamente cuando hay cambios en empleados:

registrar_alta(empleado_id, plaza_id, fecha, notas)

Se llama cuando se crea un empleado.

Si tiene plaza: estatus=ACTIVO

Si no tiene plaza: estatus=INACTIVO

registrar_asignacion(empleado_id, plaza_id, fecha, notas)

Se llama cuando se asigna un empleado a una plaza.

registrar_cambio_plaza(empleado_id, nueva_plaza_id, fecha, notas)

Se llama cuando un empleado cambia de plaza.

registrar_suspension(empleado_id, fecha, notas)

Se llama cuando se suspende un empleado. Libera su plaza.

registrar_reactivacion(empleado_id, plaza_id, fecha, notas)

Se llama cuando se reactiva un empleado suspendido.

registrar_baja(empleado_id, fecha, notas)

Se llama cuando se da de baja un empleado. Libera su plaza.

7. Integracion con empleado_service

El servicio de empleados llama automaticamente al servicio de historial cuando ocurren cambios de estado:

```
empleado_service.crear()      -> historial_service.registrar_alta()
empleado_service.dar_de_baja() -> historial_service.registrar_baja()
empleado_service.reactivar()  -> historial_service.registrar_reactivacion()
empleado_service.suspender()  -> historial_service.registrar_suspension()
```

Las llamadas al historial estan envueltas en try-except para no interrumpir el flujo principal si falla el registro del historial.

8. Interfaz de Usuario (Solo Lectura)

Estado (historial_laboral_state.py)

El estado maneja:

- Lista de registros de historial
- Filtros (estatus, busqueda)
- Modal de detalle (solo visualizacion)
- Modos de vista (tabla/cards)

Pagina (historial_laboral_page.py)

La pagina muestra:

- Banner informativo explicando que es automatico
- Filtros por estatus
- Vista de tabla o cards
- Modal de detalle (solo lectura)

NO hay boton de "Nueva Asignacion" ni acciones de edicion.

9. Errores Comunes y Soluciones

Error: VarTypeError con operador "or"

Error original:

```
VarTypeError: Cannot convert Var to bool for use with
`if`, `and`, `or`, and `not`.
```

Causa:

En Reflex, las variables de estado son reactivas (Var). No se pueden usar con operadores de Python como "or", "and", "if" directamente porque Reflex no puede convertir un Var a boolean de Python.

Codigo problematico:

```
rx.badge(tipo or "N/A", ...)  # ERROR!
```

Solucion:

Usar rx.cond() en lugar de operadores de Python:

```
rx.cond(
    tipo,                                  # condicion
    rx.badge(tipo, ...),                   # si es verdadero
    rx.badge("N/A", ...),                  # si es falso
)
```

Regla general:

- Usar rx.cond() en lugar de if/else o "or"
- Usar & en lugar de "and"
- Usar | en lugar de "or" (bitwise)
- Usar ~ en lugar de "not"

10. Flujo de Datos

Creacion de empleado:

```
1. Usuario crea empleado en UI
2. EmpleadosState.guardar_empleado()
3. empleado_service.crear()
4. empleado_repository.crear()  -> empleado guardado en BD
5. historial_service.registrar_alta()  -> registro automatico
6. historial_repository.crear()  -> historial guardado en BD
```

Suspension de empleado:

```
1. Usuario suspende empleado en UI
2. EmpleadosState.suspender_empleado()
3. empleado_service.suspender()
4. empleado_repository.actualizar()  -> estatus cambiado
5. historial_service.registrar_suspension()
   a. Cierra registro activo (fecha_fin = hoy)
   b. Crea nuevo registro con estatus=SUSPENDIDO
   c. Libera plaza del empleado
```

Visualizacion de historial:

```
1. Usuario navega a /historial-laboral
2. HistorialLaboralState.on_mount()
3. historial_service.obtener_todos()
4. historial_repository.obtener_todos()  -> query con JOIN
5. Datos mostrados en tabla/cards (solo lectura)
```

11. Estructura de Base de Datos

Tabla: historial_laboral

```
CREATE TABLE historial_laboral (
    id SERIAL PRIMARY KEY,
    empleado_id INTEGER NOT NULL REFERENCES empleados(id),
    plaza_id INTEGER REFERENCES plazas(id),  -- nullable
    tipo_movimiento estatus_tipo_movimiento,
    fecha_inicio DATE NOT NULL,
    fecha_fin DATE,  -- NULL si es registro activo
    estatus estatus_historial DEFAULT 'INACTIVO',
    notas TEXT,
    fecha_creacion TIMESTAMP DEFAULT NOW(),
    fecha_actualizacion TIMESTAMP DEFAULT NOW()
);
```

Indices importantes:

```
CREATE INDEX idx_historial_empleado ON historial_laboral(empleado_id);
CREATE INDEX idx_historial_estatus ON historial_laboral(estatus);
CREATE UNIQUE INDEX idx_historial_activo_empleado
    ON historial_laboral(empleado_id) WHERE fecha_fin IS NULL;
```