

DOCUMENTACIÓN DEL SISTEMA DE NOTIFICACIONES

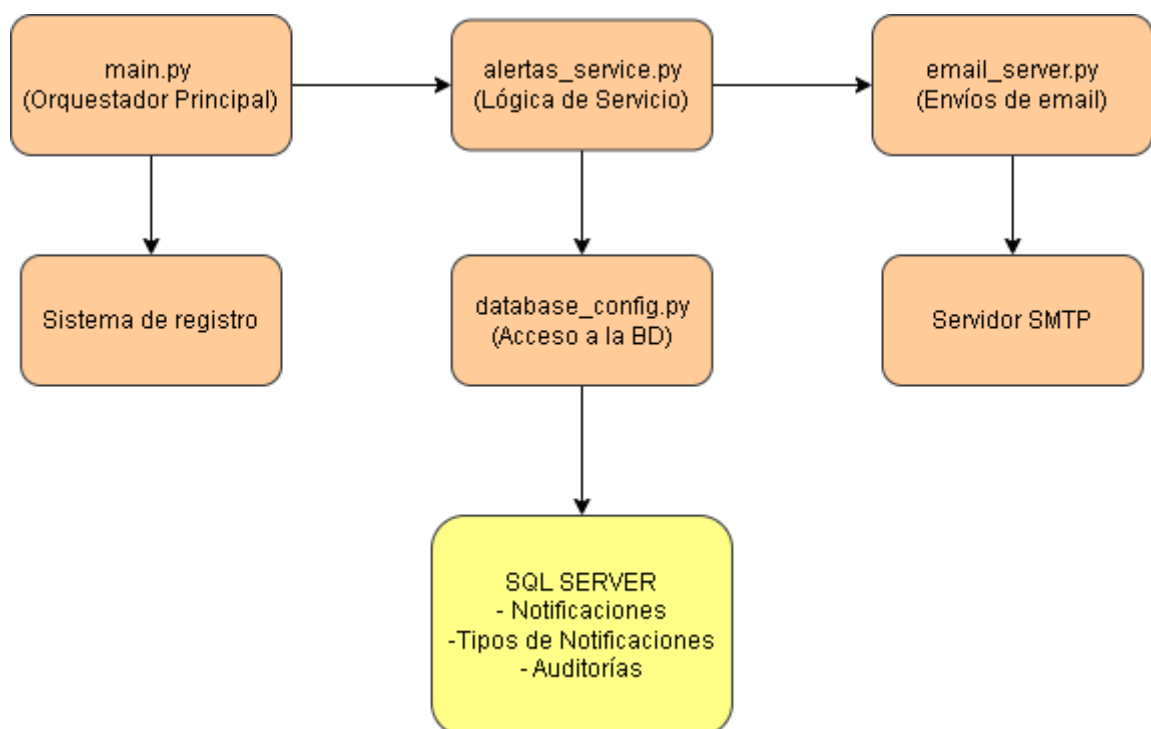
Descripción General

El **Sistema de Notificaciones** es una aplicación desarrollada en Python diseñada para **procesar, enviar y auditar notificaciones por correo electrónico de forma automatizada**. Su arquitectura modular permite una integración sencilla con bases de datos SQL Server, soporte de configuración mediante variables de entorno, y análisis visual del rendimiento a través de un dashboard interactivo creado con Plotly/Dash.

- **Procesamiento automático** de notificaciones pendientes
 - Detecta y gestiona notificaciones en estado "pendiente" desde una base de datos.
 - Aplica validaciones básicas (como la existencia de destinatario y formato de email).
- **Envío de emails** con configuración SMTP
 - Usa `smtpplib` y variables de entorno para conectarse de forma segura a un servidor SMTP.
- **Auditoría completa** de todas las operaciones
 - Registra eventos clave como envíos exitosos, errores y cambios de estado en una tabla de auditoría.
 - Trazabilidad total para cada notificación enviada o fallida.
- **Dashboard visual** para análisis
 - Visualización de métricas como la tendencia de notificaciones enviadas por tipo y la distribución de estados (enviado, pendiente, error).

Arquitectura del Sistema

Diagrama de Arquitectura



Componentes Principales

Componentes	Propósitos	Dependencias
<code>main.py</code>	Orquestador principal y bucle de ejecución	<code>alertas_service</code> , <code>logging</code>
<code>alertas_service.py</code>	Lógica procesamiento	<code>database_config</code> , <code>email_service</code>
<code>email_service.py</code>	Envío de emails vía SMTP	<code>smtplib</code>
<code>database_config.py</code>	Acceso a datos y conexión DB	<code>pyodbc</code>
<code>dashboard_plotly.py</code>	Visualización y análisis	<code>plotly</code> , <code>dash</code> , <code>pandas</code>

Componentes del Sistema

1. Main.py - Orquestador Principal

Propósito: Ejecuta el bucle principal de procesamiento cada 60 segundos.

Funcionalidades:

- Inicialización del sistema de logging
- Bucle infinito de procesamiento
- Control de intervalos de ejecución



```
1 #Flujo Principal
2 while True:
3     try:
4         ProcesadorNotificaciones.procesar_pendientes()
5     except Exception as e:
6         logger.error(f"Error en ciclo de procesamiento: {e}")
7
8     # Esperar 60 segundos entre ejecuciones
9     time.sleep(60)
```

2. AlertasService.py

Propósito: contiene la lógica principal para extraer notificaciones de la base de datos, validarlas, enviarlas y registrar auditorías.

Clase ProcesadorNotificaciones

- **Método principal:** procesar_pendientes()

Flujo de procesamiento:

1. Obtener notificaciones pendientes
2. Validar cada notificación
3. Enviar email
4. Actualizar estado
5. Registrar auditoría

Clase NotificacionesService

Métodos principales:

Método	Propósito	Parámetros	Retorno
<code>obtener_notificaciones_pendientes()</code>	Obtiene notificaciones con estado 'pendiente'	Ninguno	Lista de notificaciones
<code>actualizar_estado_notificacion()</code>	Cambia el estado de una notificación	<code>id_notificacion</code> , <code>nuevo_estado</code>	Boolean
<code>registrar_auditoria()</code>	Registra eventos en tabla de auditoría	<code>id_notificacion</code> , <code>accion</code> , <code>descripcion</code>	Boolean

3. EmailService.py - Envío de Emails

Propósito: gestiona el envío de emails utilizando configuración SMTP segura.

Configuración SMTP:

- Servidor SMTP configurable
- Autenticación con usuario/password
- Formato HTML para emails

Validaciones:

- Configuración SMTP completa
- Formato básico de email
- Manejo de errores de conexión

4. DatabaseConfig.py - Acceso a Datos

Propósito: clase reutilizable para conectarse y operar con una base de datos SQL Server utilizando `pyodbc`.

Características:

- Pool de conexiones con pyodbc
- Manejo automático de transacciones
- Conversión automática a diccionarios
- Configuración robusta con timeouts

Métodos:

- `execute_query()`: Para consultas SELECT
- `execute_non_query()`: Para INSERT/UPDATE/DELETE
- `test_connection()`: Verificación de conectividad

5. DashboardPlotly.py - Visualización

Propósito: módulo para generar visualizaciones gráficas usando plotly y dash.

Gráficos disponibles:

- **Líneas temporales:** Tendencias de notificaciones por tipo
- **Gráfico de dona:** Distribución de estados
- **Dashboard combinado:** Vista integral del sistema

Períodos soportados:

- 1 semana
- 1 mes
- 3 meses

Flujos de Proceso

Flujo Principal de Procesamiento

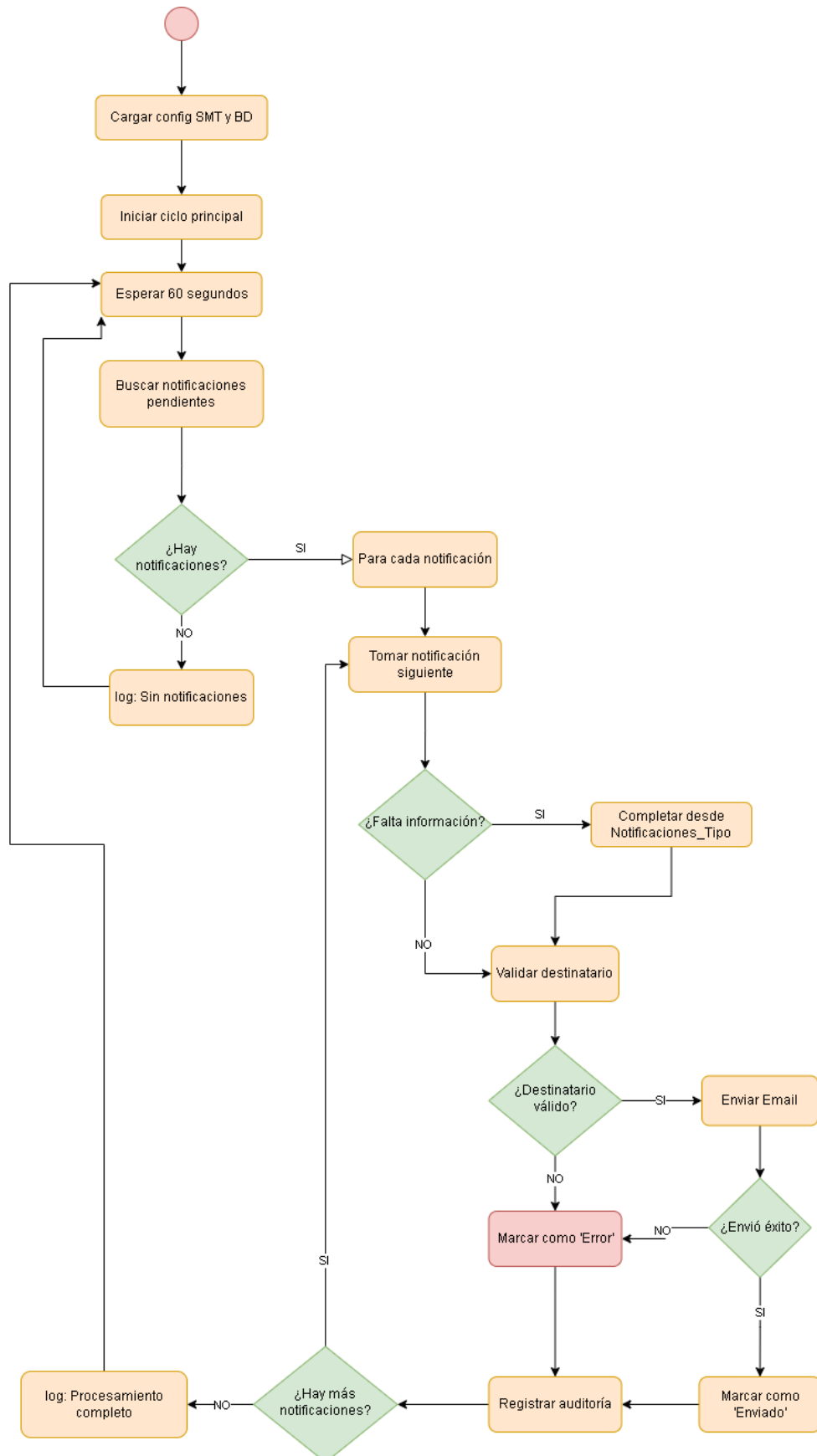
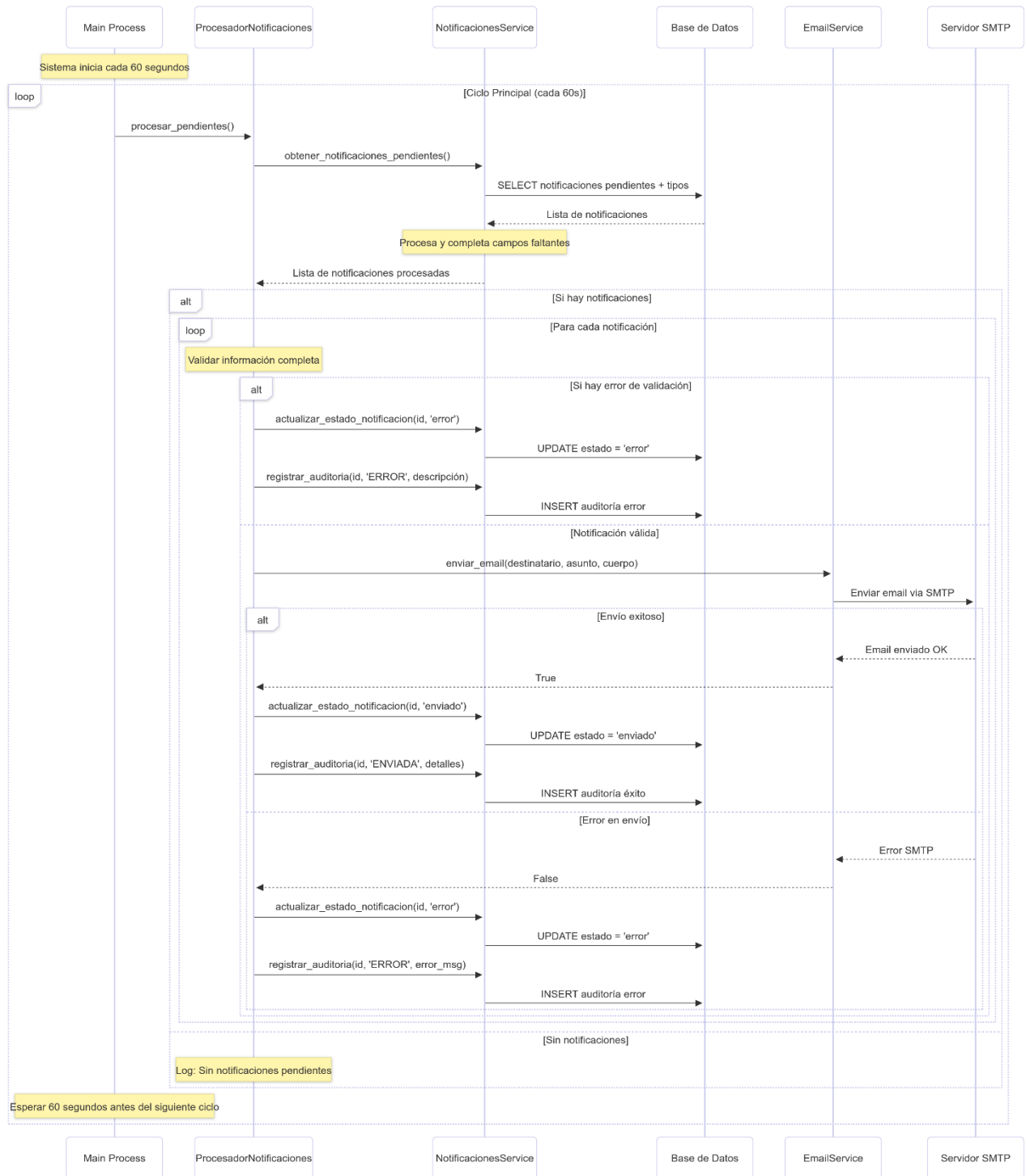


Diagrama de Secuencia - Procesamiento de Notificaciones



Base de Datos

Estructura de Tablas

Tabla: Notificaciones

```
1 CREATE TABLE Notificaciones (  
2     IdNotificacion INT IDENTITY(1,1) PRIMARY KEY,  
3     IdTipoNotificacion INT,  
4     Asunto NVARCHAR(500),  
5     Cuerpo NTEXT,  
6     Destinatario NVARCHAR(255),  
7     Estado NVARCHAR(50) DEFAULT 'pendiente',  
8     Fecha_Envio DATETIME DEFAULT GETDATE(),  
9     FOREIGN KEY (IdTipoNotificacion) REFERENCES Notificaciones_Tipo(IdTipoNotificacion)  
10 );
```

Tabla: Notificaciones_Tipo

```
1 CREATE TABLE Notificaciones_Tipo (  
2     IdTipoNotificacion INT IDENTITY(1,1) PRIMARY KEY,  
3     descripcion NVARCHAR(255),  
4     destinatarios NVARCHAR(500),  
5     asunto NVARCHAR(500),  
6     cuerpo NTEXT  
7 );  
8
```

Tabla: Auditoria

```
1 CREATE TABLE Auditoria (  
2     id INT IDENTITY(1,1) PRIMARY KEY,  
3     accion NVARCHAR(100),  
4     detalle NTEXT,  
5     fecha_aud DATETIME DEFAULT GETDATE(),  
6     [user] NVARCHAR(100) DEFAULT 'sistema'  
7 );
```

Estados de Notificaciones

Estado	Descripción	Siguiente Estado
pendiente	Notificación creada, esperando procesamiento	enviado o error
enviado	Notificación enviada exitosamente	Estado final
error	Error en el procesamiento o envío	Estado final

Uso y Ejecución

Instalación de Dependencias

```
pip install pyodbc python-dotenv smtplib logging  
pip install plotly dash pandas numpy # Para dashboard
```

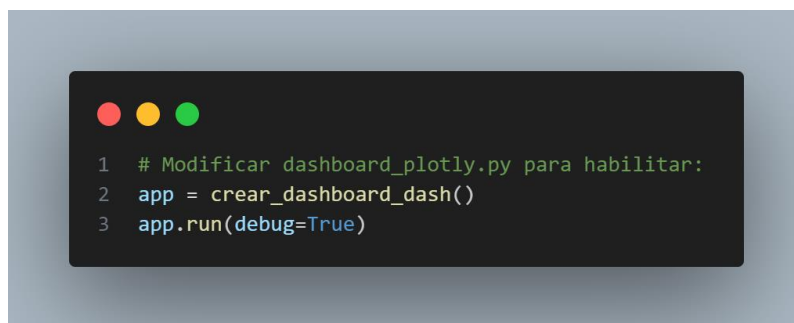
Ejecución del Sistema

1. Procesador Principal

```
python main.py
```

Comportamiento: Ejecuta continuamente, procesando notificaciones cada 60 segundos.

2. Dashboard Interactivo



Acceso: <http://localhost>

3. Dashboard Estático

```
python dashboard_plotly.py
```

Comportamiento: Genera gráficos y los muestra en el navegador.

Monitoreo y Dashboard

Métricas Disponibles

1. Tendencias Temporales

- Notificaciones por día/semana/mes
- Distribución por tipo de notificación
- Líneas de tendencia automáticas

2. Estados de Notificaciones

- Porcentaje de éxito (enviado)
- Porcentaje de errores
- Notificaciones pendientes

3. Análisis por Período

- Vista semanal (últimos 7 días)
- Vista mensual (últimos 30 días)
- Vista trimestral (últimos 90 días)

Dashboard Interactivo


Características:

- Filtros por período
- Actualización automática
- Gráficos responsivos
- Exportación a HTML

Componentes:

1. **Selector de período:** Dropdown para elegir rango temporal
2. **Gráfico de líneas:** Tendencias por tipo de notificación
3. **Gráfico de dona:** Distribución de estados actual
4. **Estadísticas resumidas:** Totales y promedios

Uso del Dashboard



```
1 # Dashboard estático simple
2 generar_dashboard_simple_plotly('1_mes')
3
4 # Dashboard interactivo
5 app = crear_dashboard_dash()
6 app.run(debug=True, port=8050)
```

Mantenimiento

Problemas Comunes

1. Error de Conexión a Base de Datos

Error: [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]Login failed

Solución: Verificar credenciales en archivo .env

2. Error SMTP

Error: (535, b'5.7.8 Username and Password not accepted')

Solución:

- Verificar credenciales SMTP
- Para Gmail: usar contraseña de aplicación
- Verificar que el servidor SMTP permita conexiones

3. Notificaciones Sin Procesar

Síntomas: Notificaciones permanecen en estado 'pendiente'

Causas posibles:

- Sin destinatarios configurados
- Email inválido
- Error en configuración SMTP