

Estándares

Se exige usar todos los estándares para una claridad y un único estilo al código, incluyendo el uso del inglés en todo el apartado de código.

Estándares a utilizar:

- * PEP 8
- * Documentación de Django
- * Conventional Commits

Implementaciones adicionales

Investigar e implementar el uso de:

```
import requests
from django.dispatch import receiver, Signal
```

Esto es necesario para la comunicación con las APIs externas ([NTFY.SH](#) y RelayHarbor) y el manejo de eventos desacoplados.

Investigar:

- * Uso de Django Signals (post_save o señales personalizadas) para disparar notificaciones sin ensuciar la lógica de los modelos principales.
- * Manejo de excepciones con requests.exceptions.RequestException para evitar que una falla en la API de correos tumbe el sistema.

Consideraciones específicas

- * Configuración Global: Crítico. Dado que el AC (Administrador del Centro) define la configuración, se recomienda usar un modelo tipo NotificationConfig o SystemSettings que actúe como "Singleton" (o limitar su creación a una única instancia).
- * Seguridad: Las credenciales de RelayHarbor y los tópicos de [NTFY.SH](#) no deben estar en texto plano en el código. Usar python-decouple o variables de entorno.
- * Roles: Validar que solo el usuario con is_superuser o rol ADMIN pueda modificar estas configuraciones.

* Manejar el control de tarea con celery

Implementación técnica

El modelo de configuración debe permitir definir los tiempos y los canales habilitados.

Atributos sugeridos (usar snake_case):

- * notification_type: ChoiceField (Ej: 'CLASS_REMINDER', 'PAYMENT_CONFIRMATION').
- * execution_time: TimeField (Para definir la hora de envío programado).
- * enable_ntfy: BooleanField.
- * enable_email: BooleanField.
- * target_roles: JSONField o ArrayField (PostgreSQL) para almacenar qué roles reciben esto (Ej: ['yogi', 'instructor']).

Relación con APIs externas:

- * [NTFY.SH](#): Método POST simple al tema configurado.
- * RelayHarbor: Configurar payload JSON estándar para el envío de correos transaccionales.

Prueba de testeo

Abrir consola:

```
python manage.py shell
```

Primero importar las librerías:

```
from tu_app.models import NotificationConfig, YogaCenterUser
from django.conf import settings
import requests
```

Prueba número 1: Crear configuración base

```
# Simulamos al Admin creando una regla de notificación
config = NotificationConfig.objects.create(
    notification_type="CLASS_CANCELLATION",
    enable_ntfy=True,
    enable_email=False, # Solo alerta rápida por ntfy
    target_roles=["yogi", "instructor"]
)
```

```
# Validamos creación
```

```
print(config.id)
# Debería salir un número (ej: 1)
print(config.enable_ntfy)
# Debería salir: True
```

Prueba número 2: Simulación de envío a NTFY (Mock)

```
# Simulamos el payload que enviaría el sistema
topic = "yoga_center_alerts"
message = "La clase de las 5:00 PM ha sido cancelada."
```

```
# Esto simula lo que hará tu función de utilidad
```

```
try:
    if config.enable_ntfy:
        resp = requests.post(
            f"https://ntfy.sh/{topic}",
            data=message.encode(encoding='utf-8')
        )
        print(f"Estado API: {resp.status_code}")
    except Exception as e:
        print(f"Error de conexión: {e}")
```

```
# Debería responder: Estado API: 200
```

Prueba número 3: Validación de Permisos (Lógica)

```
# Intentamos que un Instructor cambie la configuración global
usuario_instructor = YogaCenterUser.objects.get(username="instructor_juan")
```

```
if not usuario_instructor.is_superuser:
```

```
print("ACCESO DENEGADO: Solo el Administrador puede configurar notificaciones.")  
else:  
print("Configuración actualizada.")  
  
# Debería responder: "ACCESO DENEGADO..."
```