



# Guía Completa de Implementación - Sistema de Calificaciones



## Índice

1. Requisitos Previos
  2. Instalación Paso a Paso
  3. Configuración de Email
  4. Integración de Gráficos
  5. Testing y Verificación
  6. Despliegue
- 



## Requisitos Previos

### Dependencias a Instalar

```
bash
```

```
# Backend
```

```
pip install django==5.1
```

```
pip install celery==5.3.4
```

```
pip install redis==5.0.1
```

```
pip install django-celery-beat==2.5.0
```

```
# Frontend (si usas npm)
```

```
npm install chart.js recharts lucide-react
```

### Estructura de Archivos

```
tu_proyecto/
├── classes/
│   ├── models.py      # Modelos de calificaciones
│   ├── admin.py       # Configuración del admin
│   ├── views.py       # Vistas de calificaciones
│   ├── tasks.py       # Tareas de Celery
│   └── management/
│       └── commands/
│           └── test_email.py # Comando para testing
├── teachers/
│   ├── views.py       # Dashboard docente
│   └── templates/
│       └── teachers/
│           ├── dashboard.html      # Dashboard principal
│           └── dashboard_unified.html # Dashboard con gráficos
├── students/
│   ├── views.py       # Vistas de estudiantes
│   └── templates/
│       └── students/
│           └── student_detail.html
├── utils/
│   └── notifications.py # Sistema de notificaciones
├── templates/
│   └── emails/
│       ├── reporte_calificaciones.html
│       ├── alerta_rendimiento.html
│       └── reporte_docente.html
└── static/
    └── js/
        └── charts.js # Configuración de gráficos
```

## Instalación Paso a Paso

### PASO 1: Configurar el Modelo de Datos

#### 1.1 Agregar en `classes/models.py`

```
python
```

```
# Ver el código completo en el artifact "Sistema de Calificaciones UNIFICADO e INTEGRAL"
```

```
# Copiar desde la sección "PASO 1: Modelo Unificado"
```

#### 1.2 Crear y aplicar migraciones

```
bash
```

```
python manage.py makemigrations
```

```
python manage.py migrate
```

### 1.3 Crear tipos de aportes iniciales

```
bash
```

```
python manage.py shell
```

python

```
from classes.models import TipoAporte
```

```
# Crear tipos de aportes
```

```
TipoAporte.objects.create(  
    nombre="Trabajo en Clase",  
    codigo="TRAB_CLASE",  
    descripcion="Actividades realizadas en el aula",  
    peso=1.0,  
    orden=1  
)
```

```
TipoAporte.objects.create(  
    nombre="Exposición del Deber",  
    codigo="EXPO_DEBER",  
    descripcion="Presentación de tareas asignadas",  
    peso=1.0,  
    orden=2  
)
```

```
TipoAporte.objects.create(  
    nombre="Transcripción",  
    codigo="TRANSCRIPCION",  
    descripcion="Trabajos de transcripción musical",  
    peso=1.0,  
    orden=3  
)
```

```
TipoAporte.objects.create(  
    nombre="Deber N1",  
    codigo="DEBER_1",  
    peso=1.0,  
    orden=4  
)
```

```
TipoAporte.objects.create(  
    nombre="Deber N2",  
    codigo="DEBER_2",  
    peso=1.0,  
    orden=5  
)
```

```
TipoAporte.objects.create(  
    nombre="Deber N3",  
    codigo="DEBER_3",  
    peso=1.0,  
    orden=6  
)
```

```
peso=1.0,  
orden=6  
)
```

## PASO 2: Configurar Admin

### 2.1 Agregar en `classes/admin.py`

```
python  
  
from django.contrib import admin  
from .models import TipoAporte, CalificacionParcial, PromedioCache  
  
@admin.register(TipoAporte)  
class TipoAporteAdmin(admin.ModelAdmin):  
    list_display = ['nombre', 'codigo', 'peso', 'orden', 'activo']  
    list_editable = ['peso', 'orden', 'activo']  
    search_fields = ['nombre', 'codigo']  
    ordering = ['orden']  
  
@admin.register(CalificacionParcial)  
class CalificacionParcialAdmin(admin.ModelAdmin):  
    list_display = ['student', 'subject', 'parcial', 'quimestre', 'tipo_aporte',  
                    'calificacion', 'get_escala', 'fecha_actualizacion']  
    list_filter = ['parcial', 'quimestre', 'subject', 'tipo_aporte', 'fecha_registro']  
    search_fields = ['student__name']  
    date_hierarchy = 'fecha_registro'  
    readonly_fields = ['fecha_registro', 'fecha_actualizacion']  
  
    def get_escala(self, obj):  
        escala = obj.get_escala_cualitativa()  
        return f"{escala['codigo']} ({obj.calificacion})"  
    get_escala.short_description = 'Escala'  
  
@admin.register(PromedioCache)  
class PromedioCacheAdmin(admin.ModelAdmin):  
    list_display = ['student', 'subject', 'tipo_promedio', 'promedio', 'fecha_calculo']  
    list_filter = ['tipo_promedio', 'subject']  
    search_fields = ['student__name']  
    readonly_fields = ['fecha_calculo']
```

## PASO 3: Dashboard Docente

### 3.1 Actualizar `teachers/views.py`

python

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from classes.models import CalificacionParcial, TipoAporte, Activity
from students.models import Student
from datetime import date
```

@login\_required

```
def teacher_dashboard_view(request):
```

```
    """Dashboard unificado con sistema de calificaciones integral"""
```

```
    teacher = request.user.teacher_profile
```

```
    # MANEJO DE FORMULARIO
```

```
    if request.method == 'POST':
```

```
        action = request.POST.get('action')
```

```
        if action == 'save_calificaciones':
```

```
            student_id = request.POST.get('student_id')
```

```
            subject = request.POST.get('subject')
```

```
            parcial = request.POST.get('parcial', '1P')
```

```
            quimestre = request.POST.get('quimestre', 'Q1')
```

```
        try:
```

```
            student = Student.objects.get(id=student_id, teacher=teacher)
```

```
            tipos_aportes = TipoAporte.objects.filter(activo=True)
```

```
        for tipo in tipos_aportes:
```

```
            calificacion_value = request.POST.get(f'aporte_{tipo.id}')
```

```
            if calificacion_value:
```

```
                CalificacionParcial.objects.update_or_create(
```

```
                    student=student,
```

```
                    subject=subject,
```

```
                    parcial=parcial,
```

```
                    quimestre=quimestre,
```

```
                    tipo_aporte=tipo,
```

```
                    defaults={
```

```
                        'calificacion': float(calificacion_value),
```

```
                        'registrado_por': teacher
```

```
                    }
```

```
                )
```

```
            messages.success(request, f' Calificaciones guardadas para {student.name}')
```

```
        except Exception as e:
```

```
            messages.error(request, f' Error: {str(e)}')
```

```

return redirect('teachers:teacher_dashboard')

# DATOS DEL DASHBOARD
estudiantes = teacher.students.filter(active=True).order_by('name')
tipos_aportes = TipoAporte.objects.filter(activo=True)
materias = Activity.SUBJECT_CHOICES

# Estadísticas
estudiantes_con_promedios = []
for estudiante in estudiantes[:10]:
    promedio_general = CalificacionParcial.calcular_promedio_general(estudiante)
    estudiantes_con_promedios.append({
        'estudiante': estudiante,
        'promedio': promedio_general,
        'escala': CalificacionParcial().get_escala_cualitativa() if promedio_general > 0 else None
    })

context = {
    'teacher': teacher,
    'total_students': estudiantes.count(),
    'estudiantes': estudiantes,
    'tipos_aportes': tipos_aportes,
    'materias': materias,
    'estudiantes_con_promedios': estudiantes_con_promedios,
    'parciales': CalificacionParcial.PARCIAL_CHOICES,
    'quimestres': CalificacionParcial.QUIMESTRE_CHOICES,
}

return render(request, 'teachers/dashboard_unified.html', context)

```

### 3.2 Crear template `teachers/templates/teachers/dashboard_unified.html`

html

```
<!-- Copiar el HTML del artifact "Dashboard Docente Unificado" -->
```

## PASO 4: Sistema de Notificaciones

### 4.1 Crear `utils/notifications.py`

python

```
# Copiar el código del artifact "Sistema de Notificaciones por Email"
```

## 4.2 Crear templates de email

Archivo: `templates/emails/reporte_calificaciones.html`

html

```
<!-- Copiar el HTML del artifact "Template HTML - Reporte de Calificaciones" -->
```

## 4.3 Configurar email en `settings.py`

python

```
# Email configuration
```

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
```

```
EMAIL_HOST = 'smtp.gmail.com'
```

```
EMAIL_PORT = 587
```

```
EMAIL_USE_TLS = True
```

```
EMAIL_HOST_USER = 'tu-email@conservatorio.edu.ec'
```

```
EMAIL_HOST_PASSWORD = 'tu-app-password' # Contraseña de aplicación
```

```
DEFAULT_FROM_EMAIL = 'Conservatorio Bolívar <noreply@conservatorio.edu.ec>'
```

```
# Para testing en desarrollo
```

```
# EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

## 4.4 Obtener contraseña de aplicación de Gmail

1. Ve a <https://myaccount.google.com/security>
2. Activa "Verificación en 2 pasos"
3. Ve a "Contraseñas de aplicaciones"
4. Genera una contraseña para "Correo"
5. Copia la contraseña en `EMAIL_HOST_PASSWORD`

---

## PASO 5: Configurar Celery (Opcional - Tareas Programadas)

### 5.1 Crear `celery.py` en el directorio del proyecto



```
python

import os
from celery import Celery
from celery.schedules import crontab

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'tu_proyecto.settings')

app = Celery('tu_proyecto')
app.config_from_object('django.conf:settings', namespace='CELERY')
app.autodiscover_tasks()

# Configuración de tareas programadas
app.conf.beat_schedule = {
    'reportes-quimestrales': {
        'task': 'classes.tasks.enviar_reportes_quimestrales',
        'schedule': crontab(day_of_month='15', hour=8), # Día 15 a las 8am
    },
    'verificacion-semanal': {
        'task': 'classes.tasks.verificar_rendimiento_semanal',
        'schedule': crontab(day_of_week='monday', hour=8), # Lunes 8am
    },
}
```

## 5.2 Agregar en `settings.py`

```
python

# Celery Configuration
CELERY_BROKER_URL = 'redis://localhost:6379/0'
CELERY_RESULT_BACKEND = 'redis://localhost:6379/0'
CELERY_ACCEPT_CONTENT = ['json']
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
CELERY_TIMEZONE = 'America/Guayaquil'
```

## 5.3 Iniciar servicios

```
bash
```

```
# Terminal 1: Servidor Django
```

```
python manage.py runserver
```

```
# Terminal 2: Redis
```

```
redis-server
```

```
# Terminal 3: Celery worker
```

```
celery -A tu_proyecto worker -l info
```

```
# Terminal 4: Celery beat (scheduler)
```

```
celery -A tu_proyecto beat -l info
```

---

## Integración de Gráficos

### Opción A: Chart.js (Más Simple)

#### 1. Agregar CDN en base.html

```
html
```

```
<script src="https://cdn.jsdelivr.net/npm/chart.js@4.4.0/dist/chart.umd.min.js"></script>
```

#### 2. Crear `static/js/charts.js`

javascript

*// Gráfico de Promedios*

```
function crearGraficoPromedios(elementId, datos) {  
  const ctx = document.getElementById(elementId).getContext('2d');  
  
  new Chart(ctx, {  
    type: 'bar',  
    data: {  
      labels: datos.nombres,  
      datasets: [{  
        label: 'Promedio',  
        data: datos.promedios,  
        backgroundColor: datos.colores,  
        borderRadius: 8,  
      }]  
    },  
    options: {  
      responsive: true,  
      scales: {  
        y: {  
          beginAtZero: true,  
          max: 10  
        }  
      }  
    }  
  });  
}
```

*// Uso en template*

```
document.addEventListener('DOMContentLoaded', function() {  
  const datos = {  
    nombres: {{ estudiantes_nombres|safe }},  
    promedios: {{ estudiantes_promedios|safe }},  
    colores: {{ colores_escalas|safe }}  
  };  
  
  crearGraficoPromedios('graficoPromedios', datos);  
});
```

### 3. En la vista, preparar datos

python

```
def teacher_dashboard_view(request):
    # ... código existente ...

    # Preparar datos para gráficos
    import json

    nombres = [e.name for e in estudiantes]
    promedios = [CalificacionParcial.calcular_promedio_general(e) for e in estudiantes]
    colores = []

    for prom in promedios:
        if prom >= 9:
            colores.append('#10B981')
        elif prom >= 7:
            colores.append('#3B82F6')
        elif prom >= 4.01:
            colores.append('#F59E0B')
        else:
            colores.append('#EF4444')

    context['estudiantes_nombres'] = json.dumps(nombres)
    context['estudiantes_promedios'] = json.dumps(promedios)
    context['colores_escalas'] = json.dumps(colores)

    return render(request, 'teachers/dashboard.html', context)
```

---

## Opción B: Componente React (Más Avanzado)

### 1. Crear endpoint API en `teachers/views.py`

python

```
from django.http import JsonResponse
```

```
@login_required
```

```
def api_estadisticas(request):
```

```
    teacher = request.user.teacher_profile
```

```
    estudiantes = teacher.students.filter(active=True)
```

```
    datos = []
```

```
    for est in estudiantes:
```

```
        datos.append({
```

```
            'name': est.name,
```

```
            'promedio': float(CalificacionParcial.calcular_promedio_general(est)),
```

```
            'materias': []
```

```
        })
```

```
    return JsonResponse({'estudiantes': datos})
```

## 2. Agregar URL en `teachers/urls.py`

python

```
path('api/estadisticas/', views.api_estadisticas, name='api_estadisticas'),
```

## 3. Usar el componente React del artifact

html

```
<div id="dashboard-root"></div>
```

```
<script>
```

```
// Cargar datos reales desde API
```

```
fetch('/teachers/api/estadisticas/')
```

```
    .then(res => res.json())
```

```
    .then(data => {
```

```
        // Renderizar componente con datos reales
```

```
        ReactDOM.render(<DashboardCalificaciones data={data} />,
```

```
            document.getElementById('dashboard-root'));
```

```
    });
```

```
</script>
```



## 1. Test del Sistema de Calificaciones

```
bash
```

```
python manage.py shell
```

```
python
```

```
from students.models import Student
```

```
from classes.models import CalificacionParcial, TipoAporte
```

```
# Seleccionar estudiante
```

```
estudiante = Student.objects.first()
```

```
# Ver resumen
```

```
resumen = CalificacionParcial.obtener_resumen_estudiante(estudiante)
```

```
print(resumen)
```

```
# Calcular promedio
```

```
promedio = CalificacionParcial.calcular_promedio_general(estudiante)
```

```
print(f"Promedio: {promedio}")
```

## 2. Test de Emails

```
bash
```

```
python manage.py test_email --email=tu-email@example.com
```

## 3. Test de Gráficos

1. Acceder al dashboard docente
2. Verificar que los gráficos se renderizan correctamente
3. Comprobar que los datos son precisos



### Checklist de Pre-Despliegue

- ☐ Configurar (DEBUG = False)
- ☐ Configurar (ALLOWED\_HOSTS)
- ☐ Configurar variables de entorno para credenciales
- ☐ Recolectar archivos estáticos: (python manage.py collectstatic)
- ☐ Configurar HTTPS
- ☐ Configurar servicio de email production
- ☐ Configurar Redis en producción
- ☐ Configurar Celery como servicio systemd

## Variables de Entorno (.env)

```
bash

DEBUG=False
SECRET_KEY=tu-secret-key-segura
DATABASE_URL=postgresql://user:pass@localhost/dbname
EMAIL_HOST_USER=email@conservatorio.edu.ec
EMAIL_HOST_PASSWORD=contraseña-segura
REDIS_URL=redis://localhost:6379/0
ALLOWED_HOSTS=conservatorio.edu.ec,www.conservatorio.edu.ec
```



## Diseño Responsive

El sistema ya incluye diseño responsive con Tailwind CSS:

```
html

<!-- Grid responsive -->
<div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
  <!-- Contenido -->
</div>

<!-- Ocultar en móvil -->
<div class="hidden md:block">
  <!-- Visible solo en desktop -->
</div>

<!-- Adaptar tamaño de fuente -->
<h1 class="text-2xl md:text-4xl">Título</h1>
```



## Emails no se envían

```
python
```

```
# En settings.py, usar console backend para testing
```

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

```
# Verificar logs
```

```
python manage.py shell
```

```
from utils.notifications import NotificacionEmail
```

```
from students.models import Student
```

```
NotificacionEmail.enviar_reporte_calificaciones(Student.objects.first(), 'test@example.com')
```

## Celery no funciona

```
bash
```

```
# Verificar Redis
```

```
redis-cli ping # Debe responder "PONG"
```

```
# Ver logs de Celery
```

```
celery -A tu_proyecto worker -l debug
```

## Gráficos no se muestran

1. Verificar que Chart.js está cargado: abrir consola del navegador
2. Verificar que los datos JSON son válidos
3. Revisar errores en consola del navegador

---

## Recursos Adicionales

- [Documentación Django](#)
- [Chart.js Docs](#)
- [Celery Docs](#)
- [Tailwind CSS](#)

---

¿Necesitas ayuda? Contacta al equipo de desarrollo.