CreativeMinds AI

*Desarrollo de mi módulo para Odoo versión 17*

Heily Madelay Ajila Tandazo

Daniel Gonzalez Esteban

**Índice**

**1. Introducción**

*- 1.1. Descripción general del módulo y objetivos*

*- 1.2. Problemática que resuelve y valor añadido*

**2. Guía de Instalación**

*- 2.1. Requisitos previos y procesos de instalación paso a paso*

*- 2.2. Configuración inicial y verificación*

**3. Documentación Técnica**

*- 3.1. Arquitectura del módulo y estructura de archivos y directorios*

*- 3.2. Modelos y vistas*

*- 3.3. Seguridad y permisos*

**4. Manual de Usuario**

*- 4.1. Navegación por el módulo y funciones principales*

*- 4.2. Gestión (Proyectos, Empleados, Equipos)*

*- 4.3. Panel de control y solución de problemas frecuentes*

**5. Pruebas y Validación**

- *5.1. Pruebas realizadas*

*- 5.2. Resultados y validación*

**6. Conclusiones y Trabajo Futuro**

*- 6.1. Logros y limitaciones actuales*

*- 6.2. Propuestas de mejora a futuro*

**7. Referencias**

- *7.1. Referencias y recursos externos utilizados*

*- 7.2. Código fuente comentado y capturas de pantalla*

**1.Introducción**

**1.1. Descripción general del módulo y objetivos**

CreativeMinds AI es un módulo desarrollado para Odoo que integra herramientas de gestión de proyectos con capacidades de inteligencia artificial. Este módulo está diseñado específicamente para empresas que buscan optimizar la gestión de sus proyectos creativos y equipos de trabajo, proporcionando una interfaz intuitiva y funcionalidades avanzadas que facilitan la organización, seguimiento y control de las actividades del proyecto.

El módulo se estructura en cuatro componentes principales:

* Gestión de Proyectos
* Gestión de Empleados
* Gestión de Equipos
* Panel de Control

Tiene como objetivos principales:

* Facilitar la gestión integral de proyectos creativos
* Optimizar la asignación y seguimiento de recursos humanos
* Mejorar la colaboración entre equipos de trabajo
* Proporcionar métricas y análisis

Para alcanzar ser un sistema completo de gestión de proyectos acompañado de unas herramientas de seguimiento y control para la gestión de recursos humanos y equipos a través del análisis y reportes de rendimiento.

**1.2. Problemática que resuelve y el valor añadido**

El módulo CreativeMinds AI aborda varios desafíos comunes en la gestión de proyectos creativos:

1. **Desorganización en la gestión de proyectos:**
   1. Dificultad para seguir el progreso de múltiples proyectos
   2. Falta de centralización de la información
   3. Comunicación ineficiente entre equipos
2. **Ineficiencia en la asignación de recursos:**
   1. Sobrecarga o subutilización de personal
   2. Dificultad en la planificación de capacidades
   3. Falta de visibilidad sobre la disponibilidad de recursos
3. **Seguimiento inadecuado del progreso:**
   1. Ausencia de métricas claras
   2. Dificultad para identificar cuellos de botella
   3. Falta de transparencia en el avance del proyecto

Con este módulo aportamos el valor significativo a las organizaciones a través de la mejora en la eficiencia operativa con un mayor de control y visibilidad gracias a los dashboards en tiempo real para que los equipos puedan realizar un seguimiento detallado de proyectos para mejorar en la toma de decisiones

**2. Guía de Instalación**

**2. 1. Requisitos previos y procesos de instalación del módulo.**

Requisitos previos:

1. Tener instalado Odoo a partir de la versión 17.

2. Opcional: Tener Git instalado.

Proceso de instalación:

En caso de tener Git:

1.Abra git bash.

2.Cambia el directorio de trabajo actual a la ubicación en donde quieres clonar el repositorio (Directorio para módulos de tercero).

3.Escriba: git clone https://github.com/HeilyMadelay-hub/CreativeMinds-AI.git

4.Presione Entrar para crear el clon local.

En caso de no tener Git:

1.Entra en el repositorio: HeilyMadelay-hub/CreativeMinds-AI y bajar el .zip del repositorio.

2.Descomprimir repositorio e ir al directorio SourceCode.

3. Copia el directorio creativeminds al directorio de modulos de terceros que deseas utilizar para Odoo.

**2. 2. Configuración inicial y verificación.**

1. Modificar el archivo odoo.conf añadiendo el directorio en el que queramos trabajar con los nuevos módulos a la linea **addons\_paths**.

1- En Linux (Ubuntu, Debian, CentOS, etc.)

En sistemas basados en Linux, la ubicación más común del archivo odoo.conf es:

|  |
| --- |
| /etc/odoo/odoo.conf  /etc/odoo.conf (En algunos casos, el archivo puede estar directamente en esta ruta). |

Si no se encuentra allí, puedes buscarlo con el comando:

2- En Windows

En Windows, la ubicación del archivo de configuración puede variar dependiendo de cómo hayas instalado Odoo. Sin embargo, algunas ubicaciones comunes son:

|  |
| --- |
| C:\Program Files\Odoo\server\odoo.conf  C:\Odoo\server\odoo.conf |

Si no lo encuentras allí, puedes buscarlo en el directorio de instalación de Odoo.

3- En macOS

En macOS, si has instalado Odoo a través de un gestor de paquetes o mediante instalación manual, el archivo odoo.conf generalmente se encuentra en:

|  |
| --- |
| /etc/odoo.conf  /usr/local/odoo/odoo.conf |

Una vez configurado el archivo reinicia el servicio de odoo:

|  |
| --- |
| sudo systemctl restart odoo |

Creamos y/o accedemos al directorio (en este ejemplo se llama custom\_addons y fue creado dentro del directorio de Odoo).

- Recuerda copiar el módulo desde el directorio SourceCode o crear un zip del módulo.

**2.3. Instalación y ejecución de la API**

1. Copia el directorio creative minds\_api al lado del directorio creativeminds y muevete a él.

1.1. Ahora puedes crear un nuevo entorno virtual o acceder al de Odoo.

|  |
| --- |
| Source nombre\_entorno/bin/activate |

2. Instala las dependencias de la API, estas son: werkzeug, itsdangerous, click, flask-cors, pandas, numpy, sqlalchemy, python-dotenv y requests:

|  |
| --- |
| pip install werkzeug jinja2 itsdangerous click flask-cors pandas numpy sqlalchemy python-dotenv requests |

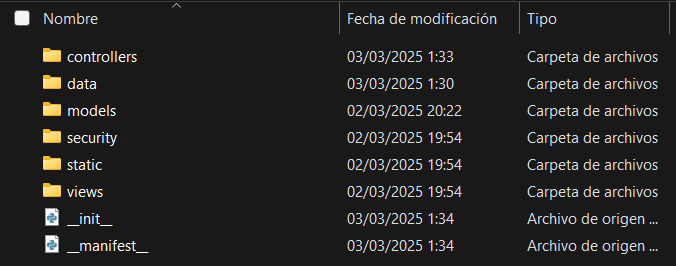
3. Entra en el directorio del api y ejecuta app.py:

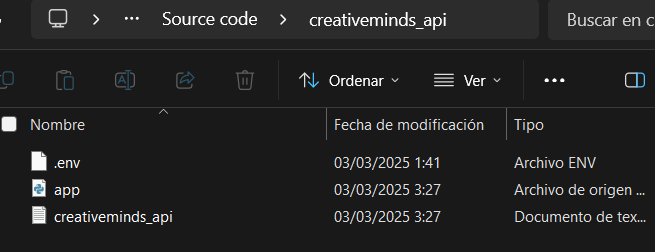
|  |
| --- |
| python app.py |

**3. Documentación Técnica**

**3.1. Arquitectura del módulo y estructura de archivos y directorios.**

Este módulo sigue la arquitectura estándar de Odoo, que se organiza en una estructura modular con el mantenimiento de la extensibilidad. Siguiendo el patrón Modelo-Vista-Controlador, donde la lógica de negocio, la presentación y el control de flujo están claramente separados.





**Dependencias:**

El módulo depende de los siguientes módulos estándar de Odoo:

base: funcionalidades básicas de Odoo.

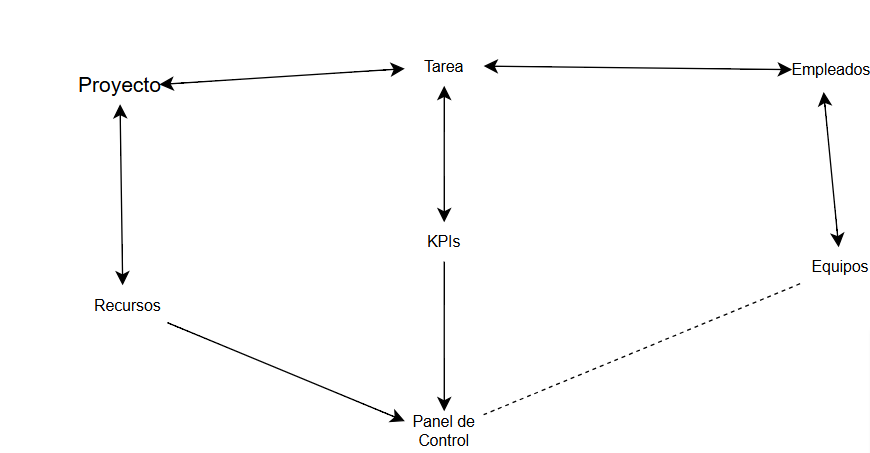
mail: sistema de comunicación y seguimiento

project: funcionalidades para la gestión de proyectos

hr: gestión de recursos humanos

**Flujo de datos:**

El diagrama de flujo de datos ilustra la estructura relacional del sistema CreativeMinds, donde las entidades principales —Proyectos, Tareas y Empleados— se interconectan bidireccionalmente, permitiendo que los proyectos contengan tareas, las tareas sean asignadas a empleados y los proyectos tengan personal asignado. Cada entidad principal se vincula con una entidad secundaria específica: los Proyectos se asocian con Recursos, las Tareas con KPIs y los Empleados con Equipos. Todas estas entidades secundarias convergen finalmente en el Panel de Control, que funciona como centro neurálgico para la visualización y monitorización integral de todos los componentes del sistema.



**3.2. Modelos y Vistas.**

**Modelos:**

* **Proyecto:**
  + Campos básicos: proyecto\_id, nombre, costo\_por\_hora, horas\_asignadas, descripcion, cliente, estado, fecha\_inicio, fecha\_fin, prioridad, presupuesto\_estimado, imagen\_proyecto, imagen\_filename, documentacion\_tecnica, documentacion\_filename, colaboradores, riesgos, hitos, dependencias, comentarios,recordatorios\_automaticos.
  + Campos calculados: costo\_total. porcentaje\_progreso, costo\_total\_recursos.
  + Campos relacionales: empleado\_id, responsable\_id, recursos\_id, tareas\_ids. indicadores\_ids, archivos\_adicionales.
  + Restricciones:\_verificar\_costo\_y\_horas,\_verificar\_presupuesto, \_verificar\_fechas\_proyecto, \_verificar\_presupuesto\_estimado, \_verificar\_campos\_importantes, \_verificar\_campos\_planificacion, \_verificar\_recursos\_minimos, \_verificar\_fechas\_y\_tareas.
  + Funciones: enviar\_notificacion\_proyecto, enviar\_recordatorio, obtener\_resumen\_proyecto, duplicar\_proyecto, ver\_tareas, ver\_recursos, ver\_miembros.
* **Recurso:**
  + Campos básicos: nombre, costo\_por\_hora, horas\_asignadas, fecha\_inicio, fecha\_fin, estado.
  + Campos relacionales: proyecto\_id, empleado\_id.
  + Campos calculados: costo\_total.
* **Tarea:**
  + Campos básicos: nombre, descripcion, fecha\_inicio, fecha\_fin, estado.
  + Campos relacionales: proyecto\_id, responsable\_id.
  + Restricciones: \_verificar\_fechas\_tarea.
* **KPI:**
  + Campos básicos: nombre, valor, objetivo.
  + Campos relacionales: proyecto\_id.
* **Empleado:**
  + Campos básicos: empleado\_id, name, dni, apèllido1, apèllido2, estado, fecha\_nacimiento, fecha\_incorporacion, foto, departamento, puesto, disponibilidad.
  + Campos relacionales: partner\_id, proyecto\_id, equipo\_id.
  + Restricciones:\_check\_dni,\_check\_edad\_minima, \_sql\_constraints..
* **Empleado:**
  + Campos básicos: equipo\_id, nombre.
  + Campos calculados: n\_mienbros.
  + Campos relacionales: empleado\_id, responsable\_id.
* **DashboardMetricas:**
  + Campos básicos: fecha\_actualizacion, proyectos\_en\_progreso, proyectos\_finalizados proyectos\_retrasados, progreso\_promedio, presupuesto\_total, costo\_actual\_total, eficiencia\_presupuestaria, total\_tareas, tareas\_completadas, tareas\_pendientes, empleados\_disponibles.
  + Campos relacionales: panel\_id.
* **Recomendaciones:**
  + Campos básicos: descripcion, prioridad, fecha, estado, notas.
  + Campos relacionales: panel\_id.
* **DashboardMetricas:**
  + Campos básicos: nombre, configuracion, fortalezas, debilidades, oportunidades, amenazas, ultima\_actualizacion.
  + Campos relacionales: proyectos\_id.
  + Funciones: \_ensure\_default\_record, \_init\_dashboard, load\_data\_from\_api

**Vistas:**

* **Proyectos: F**ormulario, lista(tree), kanban..
* **Control Panel:** Lista(tree), dashboard(formulario).
* **Empleados:** Lista(tree), formulario.
* **Equipos:** Lista(tree).
* **Tareas:** Lista(tree).
* **Recursos:** Lista(tree).

**3. 3. Seguridad y permisos.**

La seguridad en el módulo se gestiona mediante un sistema de control de acceso basado en roles implementado a través de registros y definición de grupos.

* **Registros de Acceso (ir.model.access.csv):**

El módulo define permisos para cada modelo, estableciendo diferentes niveles de acceso según el rol del usuario en la organización:

* + **Usuarios Básicos (base.group\_user):**
    - Tienen permisos para leer, crear y modificar registros, pero no pueden eliminarlos.
    - Este nivel de permisos se aplica a modelos operativos como Proyectos, Recursos, Tareas y KPIs.
    - Para modelos más sensibles como Empleados, Equipos, Horas Trabajadas y Panel de Control, únicamente se otorgan permisos de lectura.
  + **Gestores de Proyecto (project.group\_project\_manager):**
    - Disponen de permisos completos (lectura, escritura, creación y eliminación) sobre la mayoría de los modelos.
    - Controlan Proyectos, Recursos, Tareas, KPIs, Equipos, Horas Trabajadas y Panel de Control.
  + **Gestores de Recursos Humanos (hr.group\_hr\_manager):**
    - Tienen control completo sobre la información de Empleados.
    - Pueden crear, modificar y eliminar registros relacionados con el personal.
* **Grupos de Usuarios:**

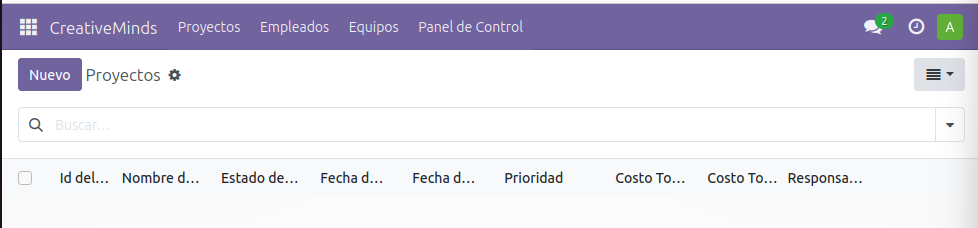
El sistema de seguridad se basa en tres grupos principales de Odoo:

* **base.group\_user:** Usuarios estándar de la plataforma. Acceden a información operativa del día a día con restricciones en operaciones críticas.
* **project.group\_project\_manager:** Responsables de la gestión de proyectos. Tienen control total sobre los recursos, tareas y elementos relacionados con proyectos.
* **hr.group\_hr\_manager:** Encargados de recursos humanos. Manejan la información sensible relacionada con empleados.

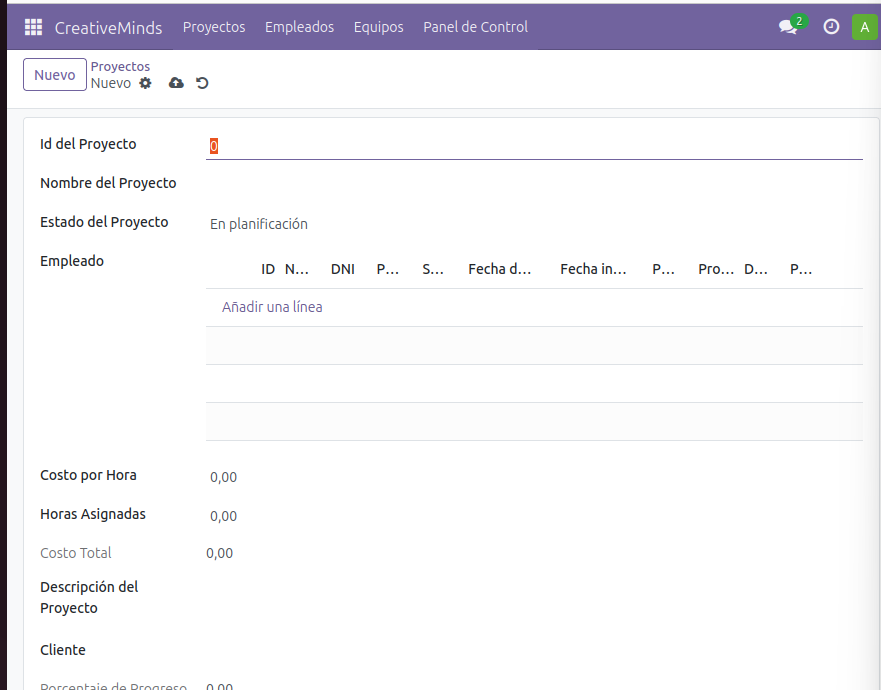
**4.Manual de Usuario**

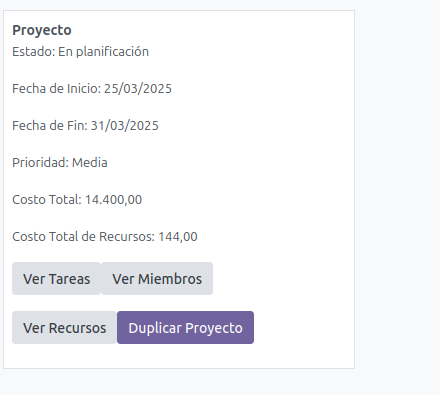
**4.1. Navegación por el módulo y funciones principales**

Nada mas entrar nos encontramos con “el protagonista del módulo”, Proyectos:

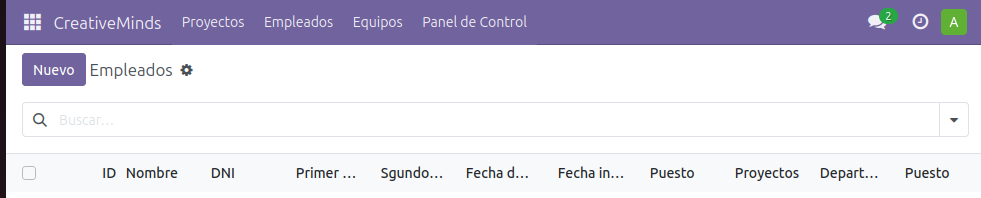


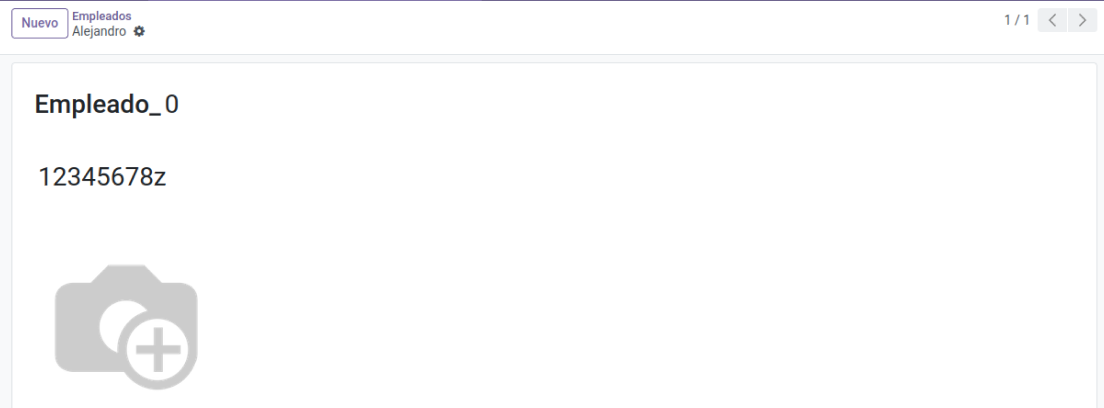
Vamos a proceder a crear un proyecto:

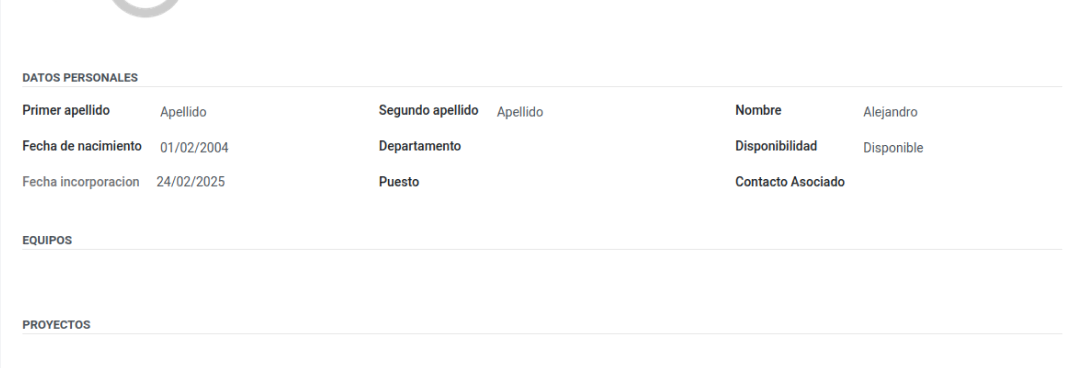


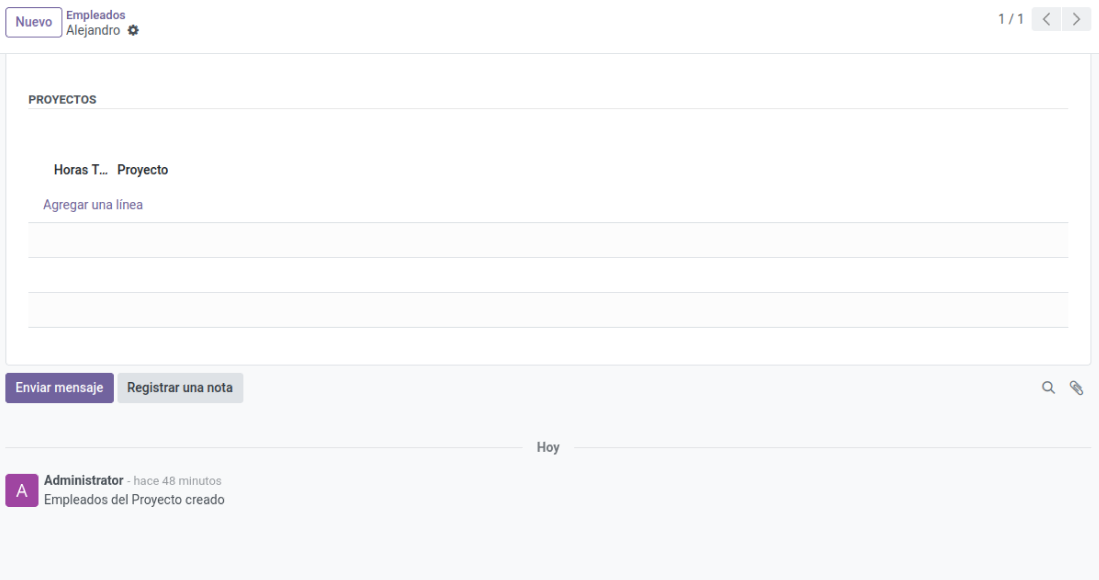


La clase Empleado, donde podremos crear un nuevo Empleado:

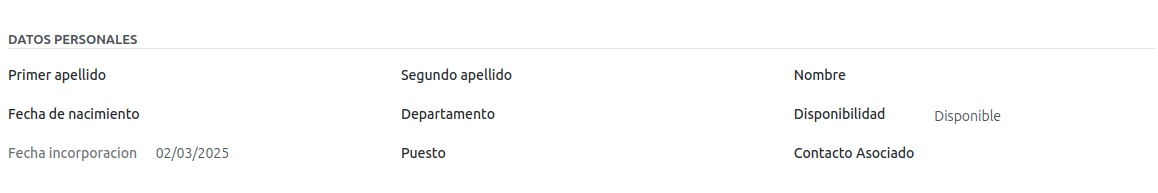




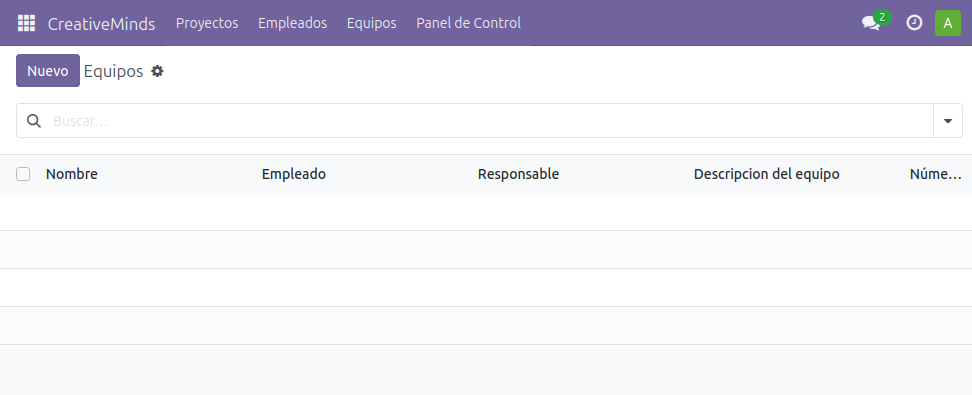


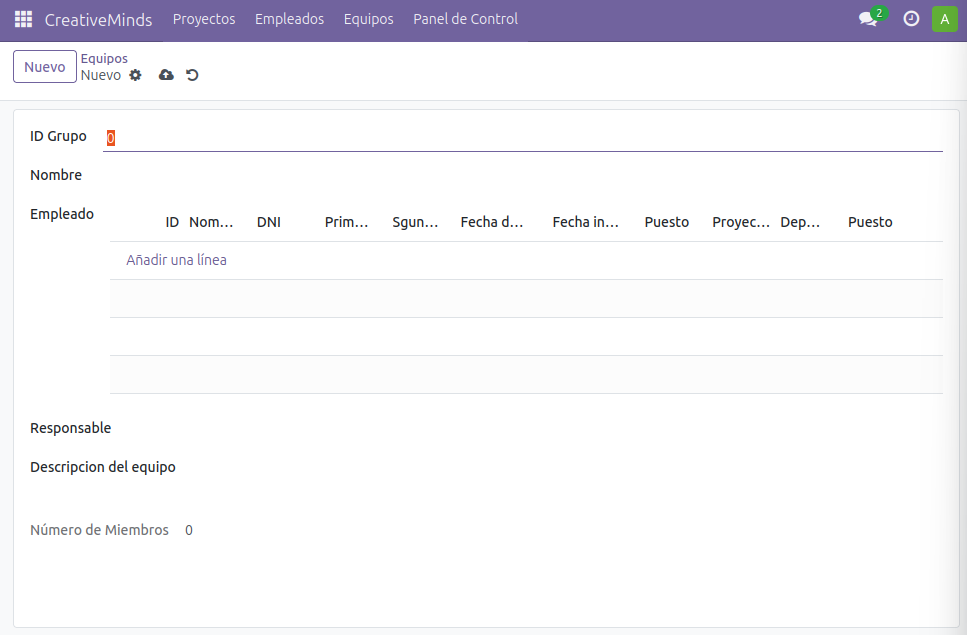


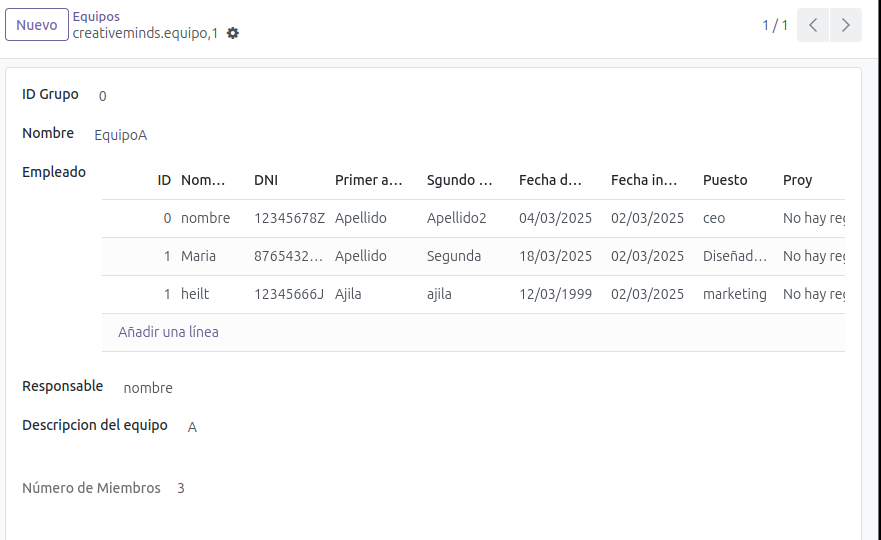
[El formulario corregido]:



La clase Equipos, donde podremos crear un nuevo Equipo para el proyecto:







Creamos varios datos de Proyecto, Empleados y Equipos para pasar a mostrar Panel Control.

**4.2. Gestión (Proyectos, Empleados, Equipos)**

**Proyectos**:

El módulo permite gestionar el ciclo completo de **proyectos**, integrando todos los aspectos necesarios para su ejecución efectiva. Cada **proyecto** se define con un **ID único**, un **nombre** y una **descripción**, lo que facilita su identificación y gestión. Se asignan **recursos** y **empleados** a los **proyectos** para distribuir adecuadamente las responsabilidades y tareas.

El **costo total** del proyecto se calcula automáticamente en función del **costo por hora** de los empleados y las **horas estimadas** para completarlo, lo que ofrece una visión clara de los costos y permite validar el **presupuesto** a lo largo del ciclo del proyecto. Además, se realiza un seguimiento de los **presupuestos estimados** versus los **costos reales**, asegurando que no se exceda el presupuesto.

El progreso de cada **proyecto** se gestiona mediante **estados configurables** como planificación, en progreso, finalizado o detenido, y el **porcentaje de avance** se calcula automáticamente en función de las tareas completadas. El sistema envía **notificaciones automáticas** para mantener informados a todos los involucrados sobre cualquier cambio en el estado o progreso del proyecto.

El módulo también permite una **gestión detallada de las tareas** asociadas al proyecto, incluyendo responsables, fechas de inicio y finalización, y estado. Los **recordatorios automáticos** garantizan que las **tareas** se completen a tiempo.

Además, se pueden adjuntar **documentos** e **imágenes** al proyecto, asegurando que toda la **documentación** necesaria esté disponible para el equipo y los interesados. Se realiza un seguimiento de los **recursos** asignados al proyecto, con un control de **costos**, y se validan las **fechas de inicio** y **finalización** para evitar discrepancias.

Entre las funcionalidades adicionales, se incluye la posibilidad de **duplicar proyectos** junto con las **tareas** y **recursos** asociados, así como la generación de **resúmenes detallados** sobre el progreso, las **tareas** y el **presupuesto**, lo que brinda una visión completa del estado del proyecto en todo momento.

**Empleado:**

El módulo permite gestionar la información de los **empleados**, su **disponibilidad** y las **tareas** que tienen asignadas. En él se guardan datos como el **ID**, **nombre**, **DNI**, apellidos y **fecha de nacimiento**. Se asegura que el **DNI** esté en el formato correcto y que los **empleados** tengan al menos 16 años. Además, se controla si el **empleado** está **disponible**, asignado a algún proyecto, parcialmente disponible o no disponible.

Cada **empleado** puede estar relacionado con **proyectos** y **equipos** dentro del módulo, lo que facilita conocer en qué **proyectos** y **equipos** está trabajando cada persona. Además, al crear un nuevo **empleado**, el sistema envía una **notificación automática** para informar a las personas responsables sobre el nuevo registro.

En cada registro se mostrarán los **datos personales** del **empleado**, los **equipos** con los que trabaja, los **proyectos** en los que participa y un **historial de cambios** realizados. Además, se verifica que el **DNI** del **empleado** sea único y que tenga la edad mínima para poder trabajar.

**Equipo:**

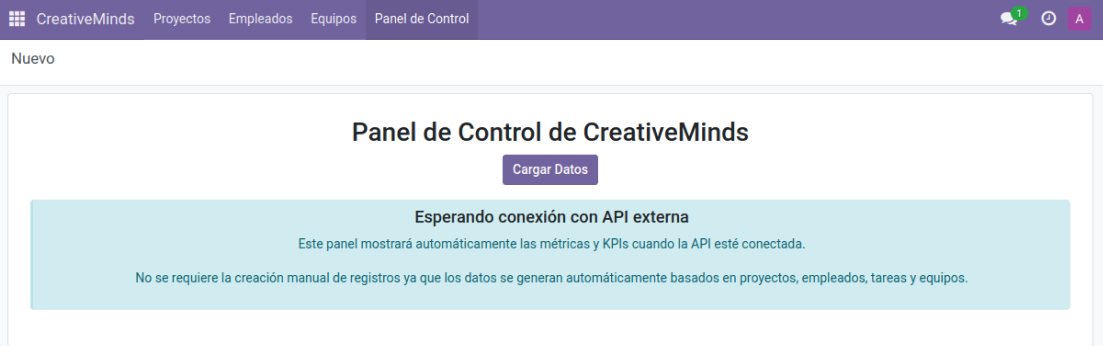
El módulo permite gestionar los **equipos de trabajo** dentro de la empresa. Cada **equipo** tiene un **ID único**, un **nombre** y una **descripción**, lo que ayuda a identificar y comprender la función de cada grupo.

Cada **equipo** tiene **empleados** asignados, lo que facilita saber quiénes forman parte de él. Además, se asigna un **responsable** o líder, que es la persona encargada de coordinar y supervisar al grupo.

El sistema calcula automáticamente cuántos miembros tiene cada **equipo** y muestra este número en el campo **n\_miembros**, actualizándose de manera automática cada vez que se agregan o eliminan **empleados** del **equipo**.

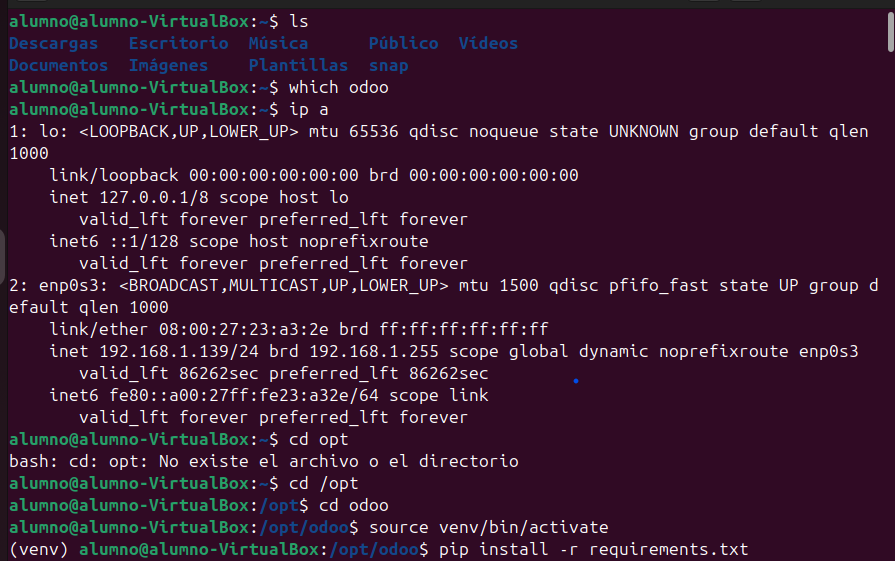
En resumen, el módulo de **equipos** ayuda a organizar y gestionar los grupos de trabajo, mostrando quién está en cada **equipo**, quién es el **responsable** y cuántos **miembros** componen cada grupo.

**4.3. Panel de control y solución de problemas frecuentes**

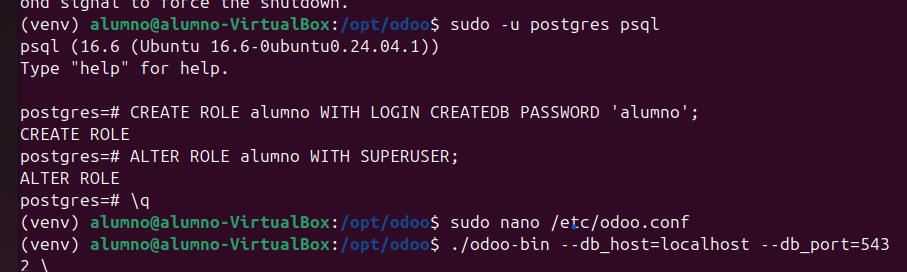


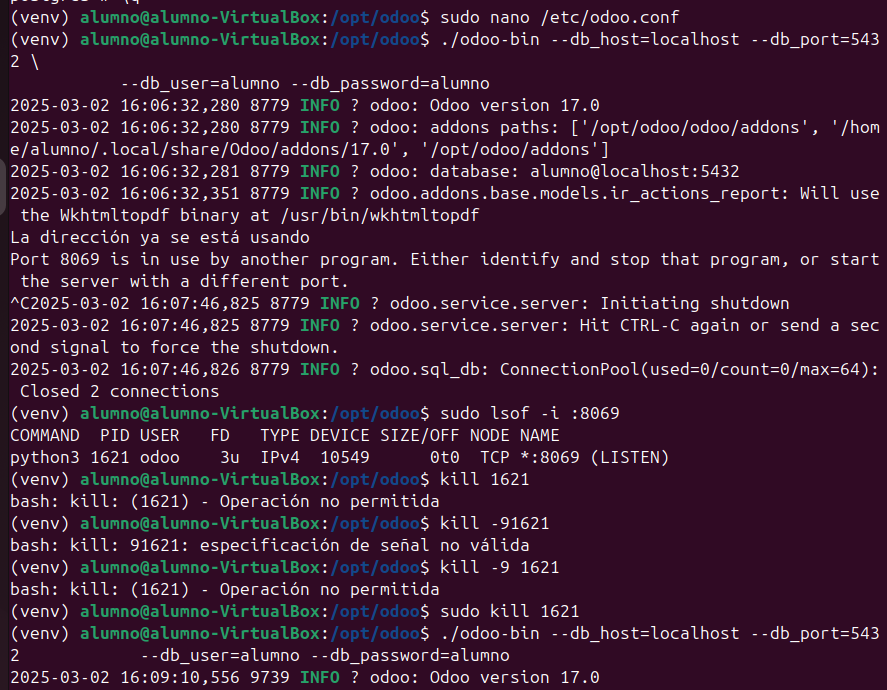
**5.Pruebas Realizadas y Resultados Validación**

Prueba de instalación:

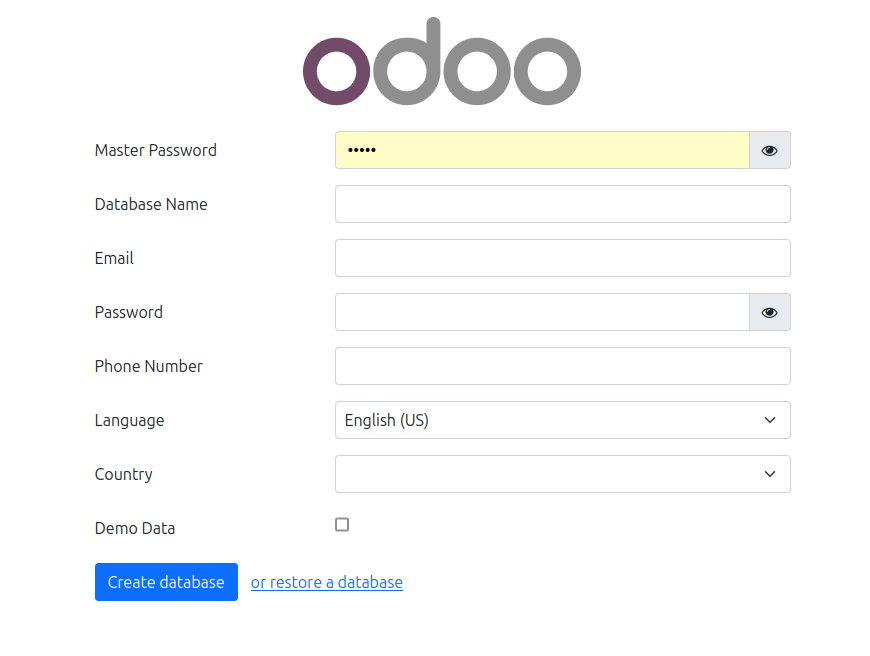


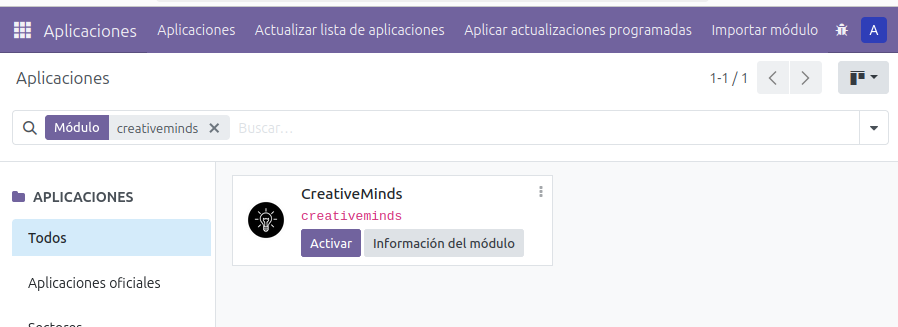


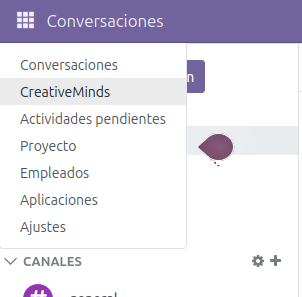




Luego después del último comando, nos mandara a la siguiente imagen para crear la base de datos:´

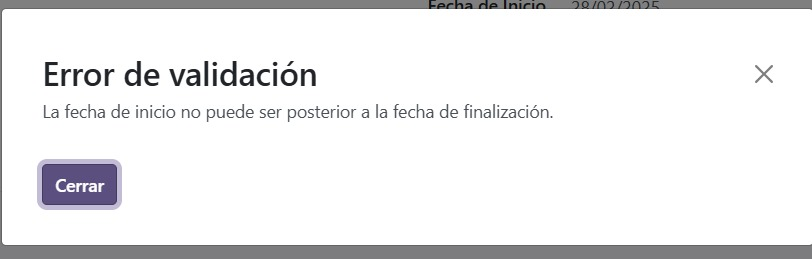


Vamos: Ajustes->Activamos el modo desarrollador->Aplicaciones->Actualizar lista de aplicaciones->Activar

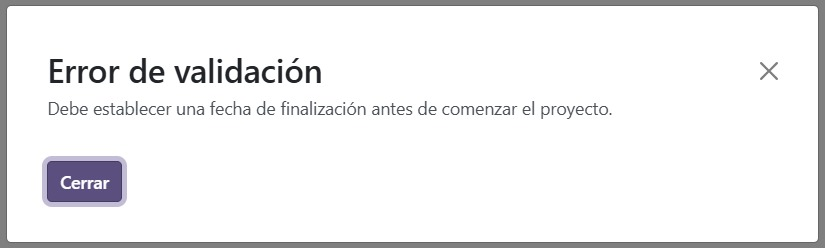
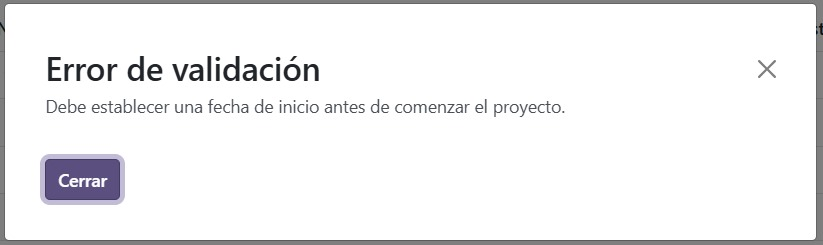


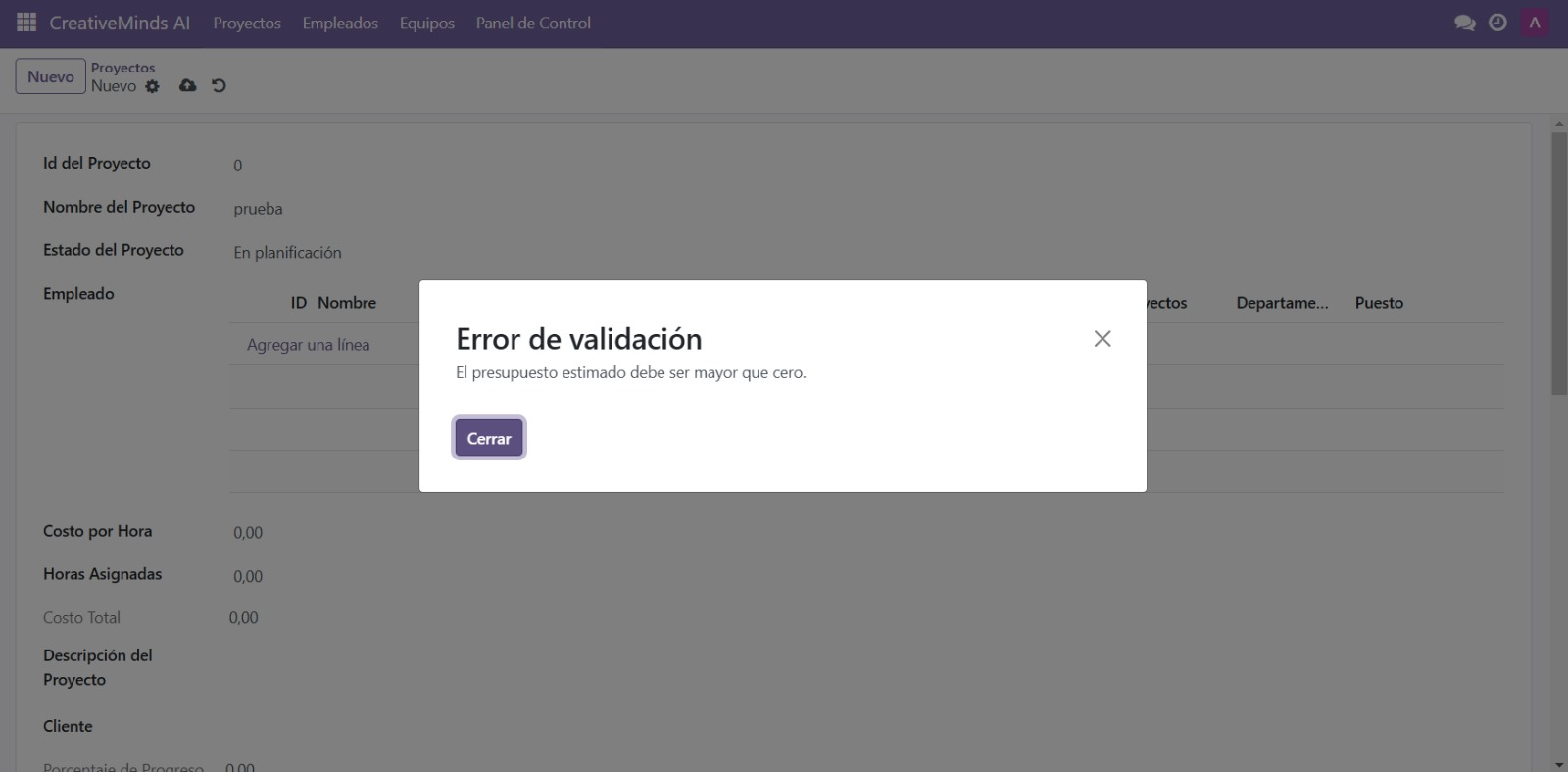
Prueba de validación de fechas

Caso 1: Intentamos crear un proyecto con una fecha de inicio posterior a la de finalización

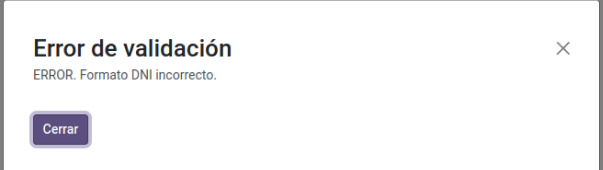


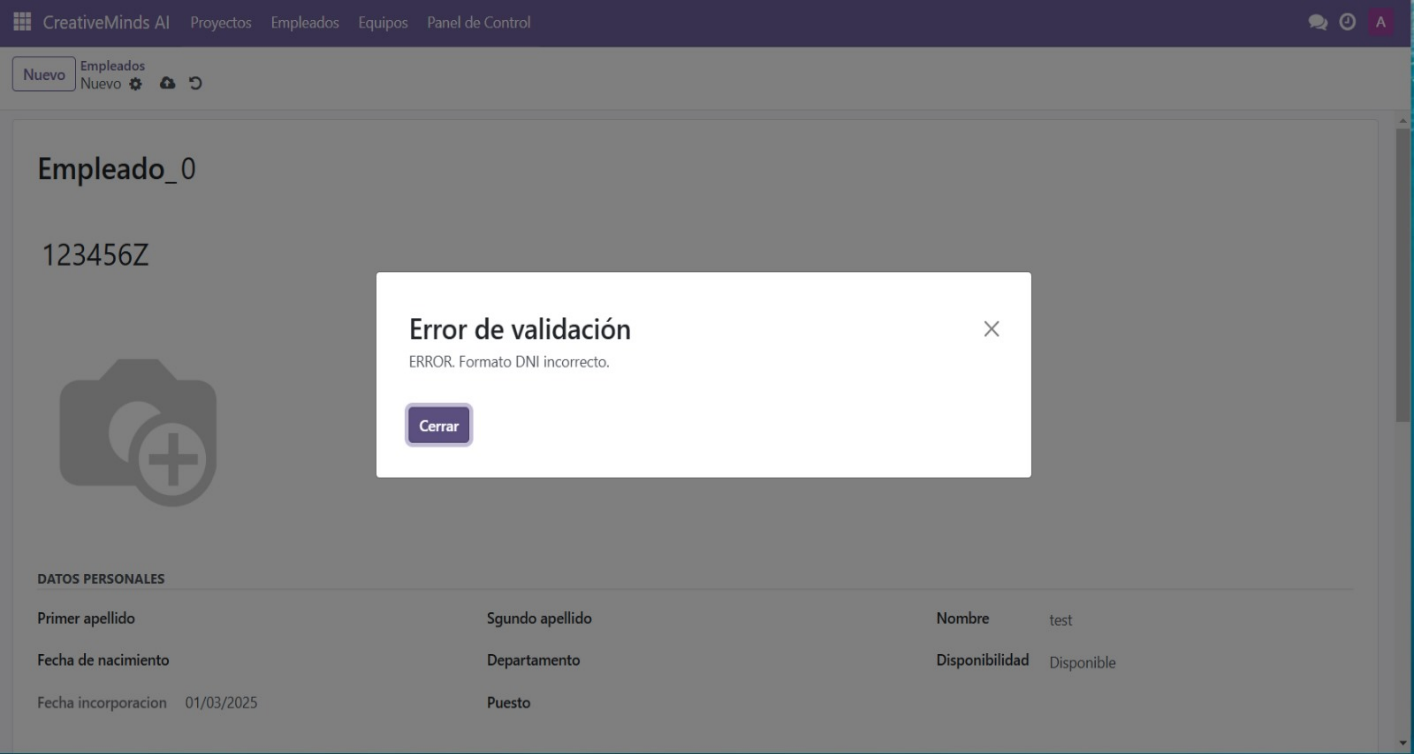
Caso 2: Intentamos crear un proyecto sin fecha de inicio/finalización.



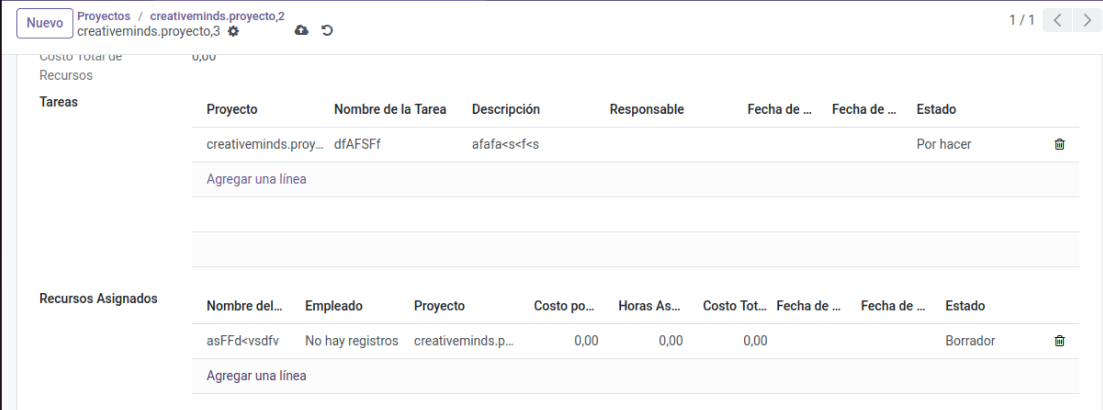
Prueba de validación de presupuesto

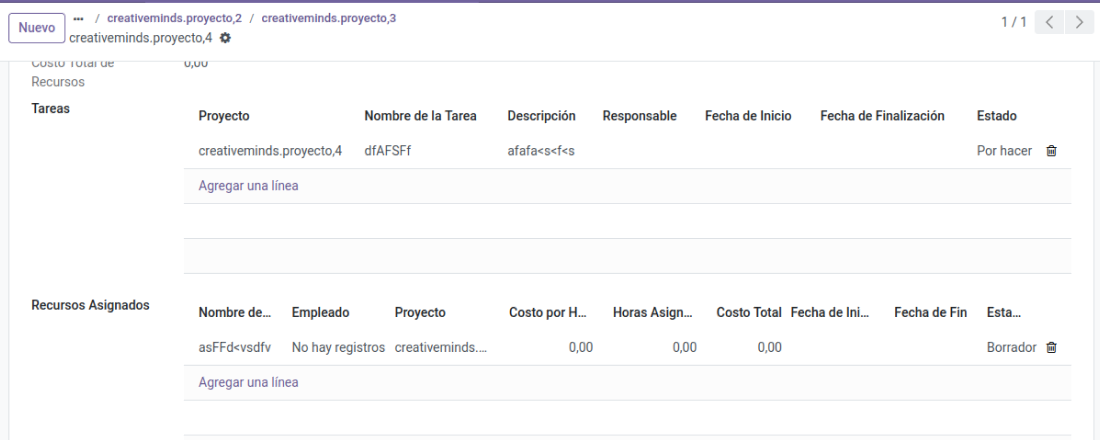
Prueba de validación de DNI



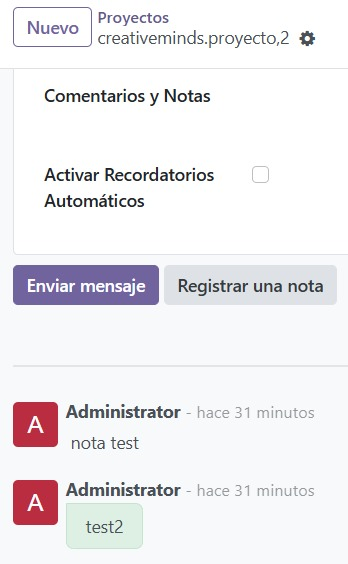
Caso: Creamos un empleado con un DNI incorrecto

Prueba de duplicación de proyecto

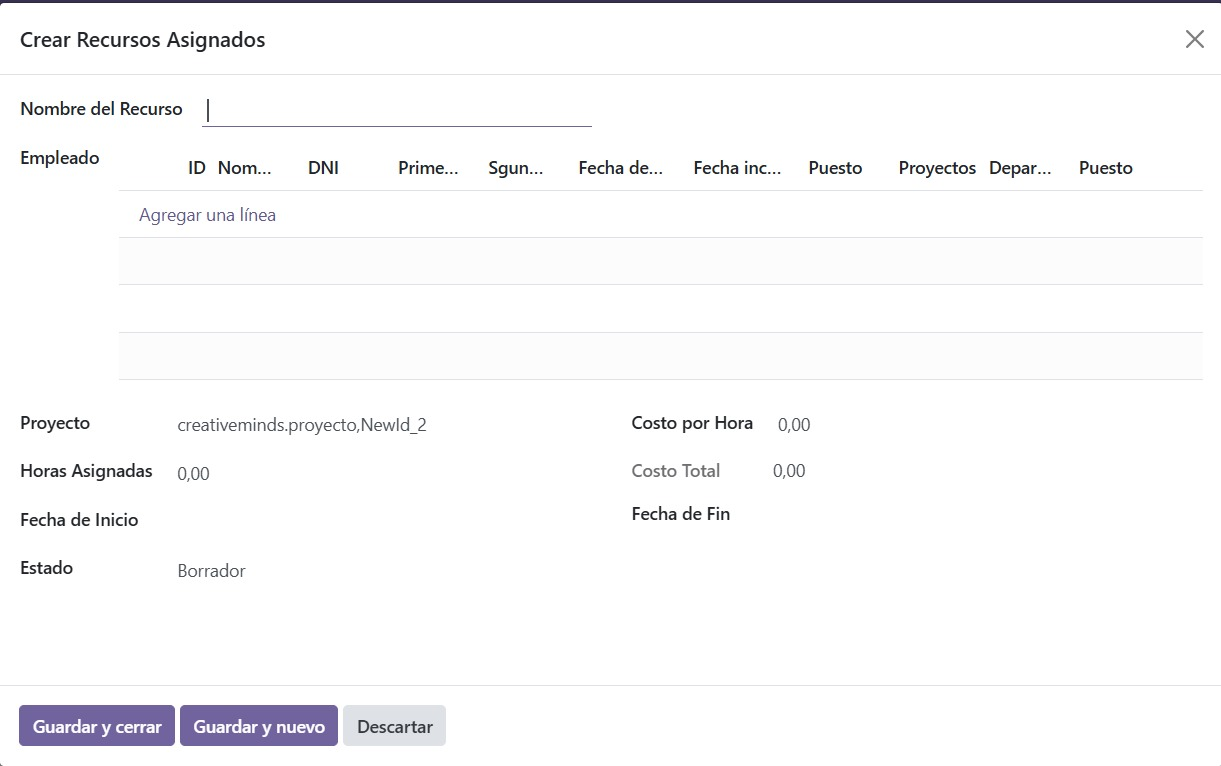




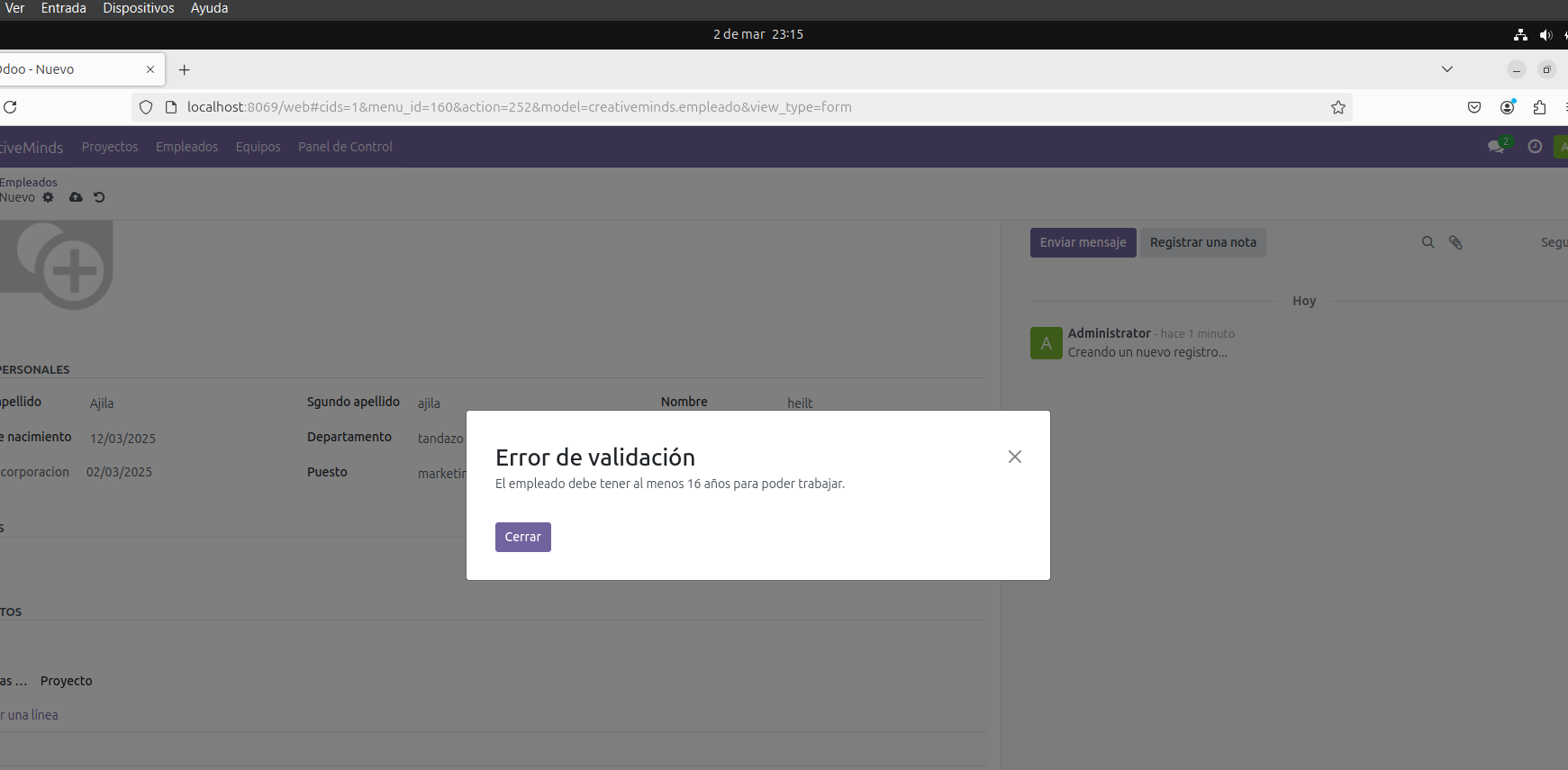
Prueba de envío de notificaciones:

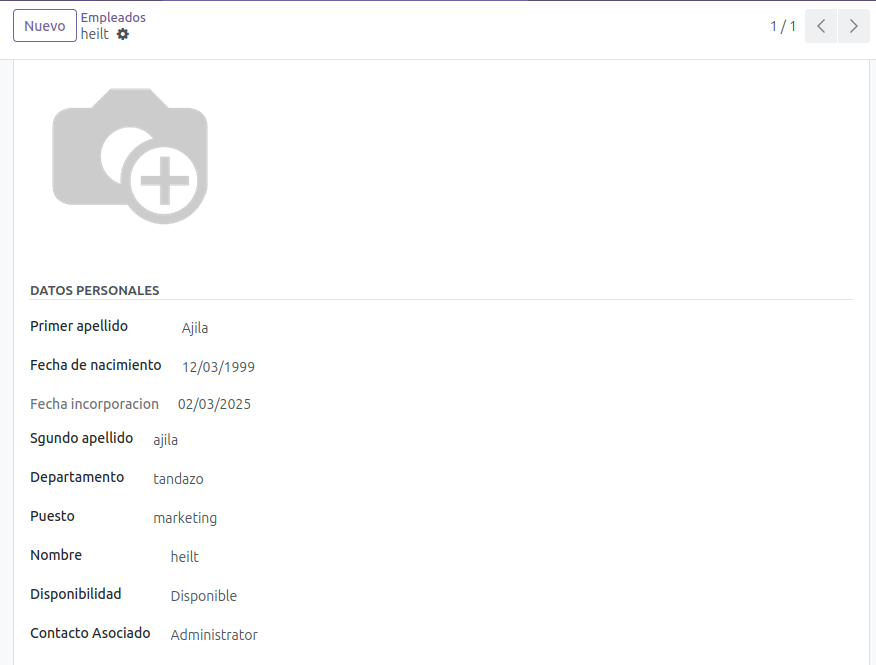


Prueba de creación de recursos:

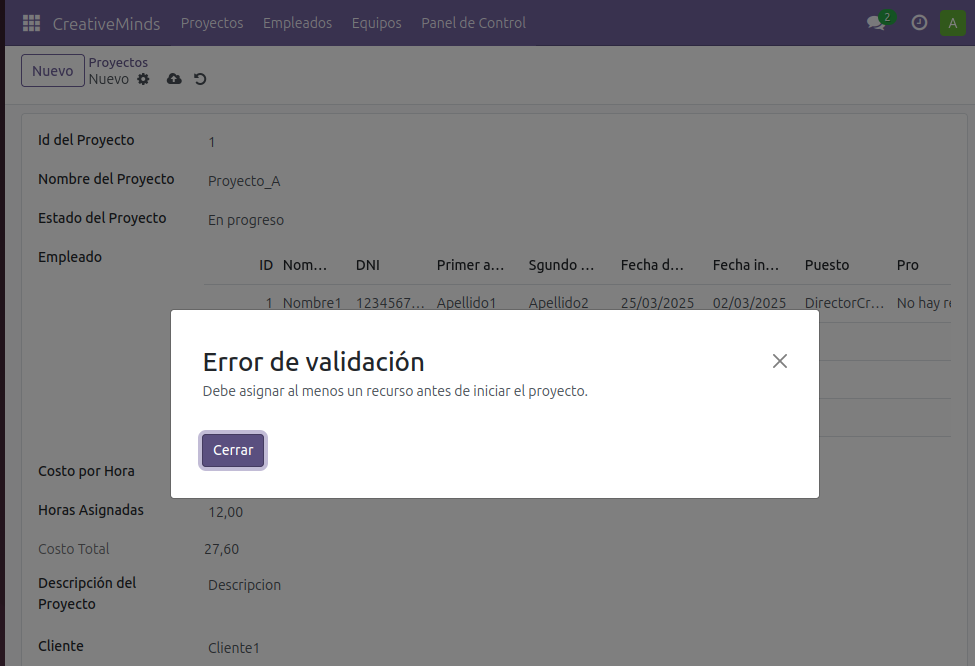


Prueba: No puede trabajar cualquier persona menos de 16 años:



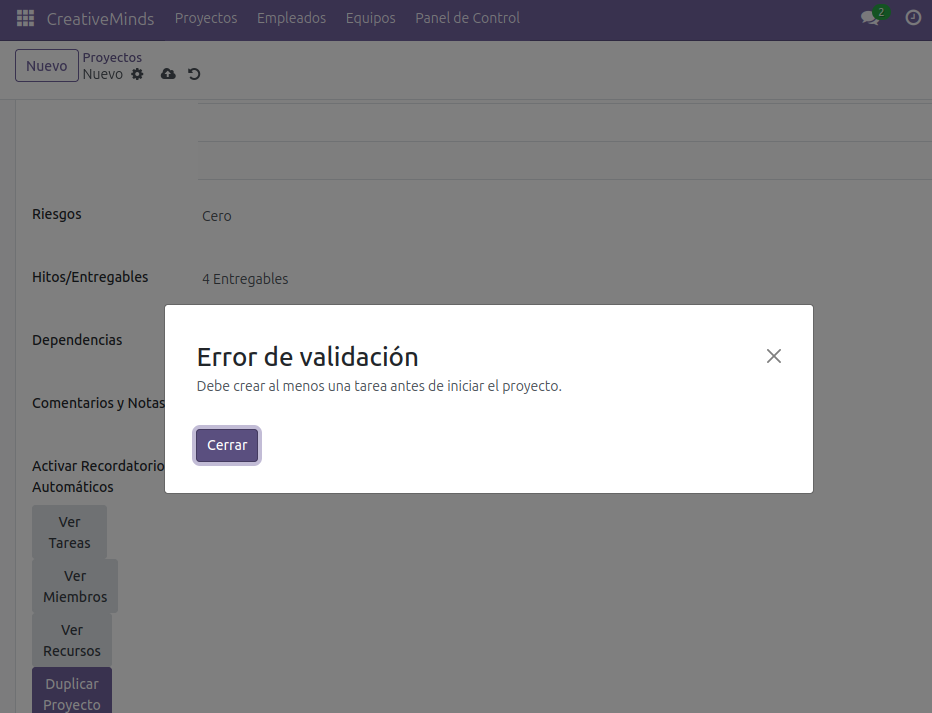


Prueba: Intentas crear un proyecto pero sin asignarle recursos:

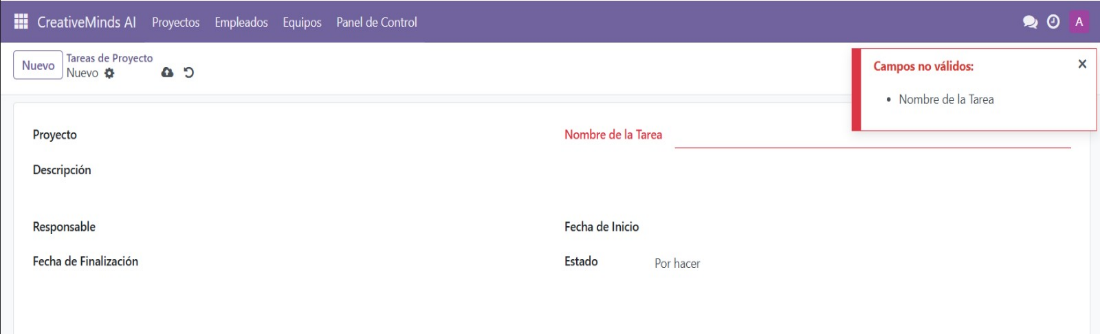


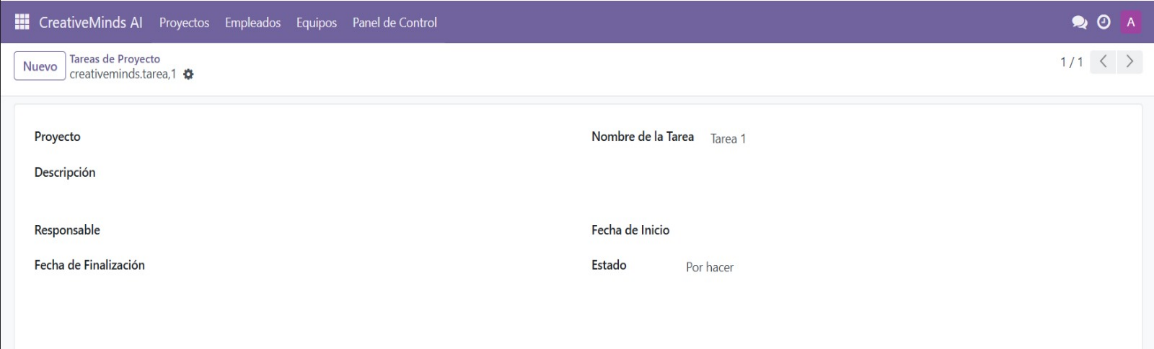
Prueba de creación de tareas:

Caso: Intentamos crear un proyecto sin tarea:



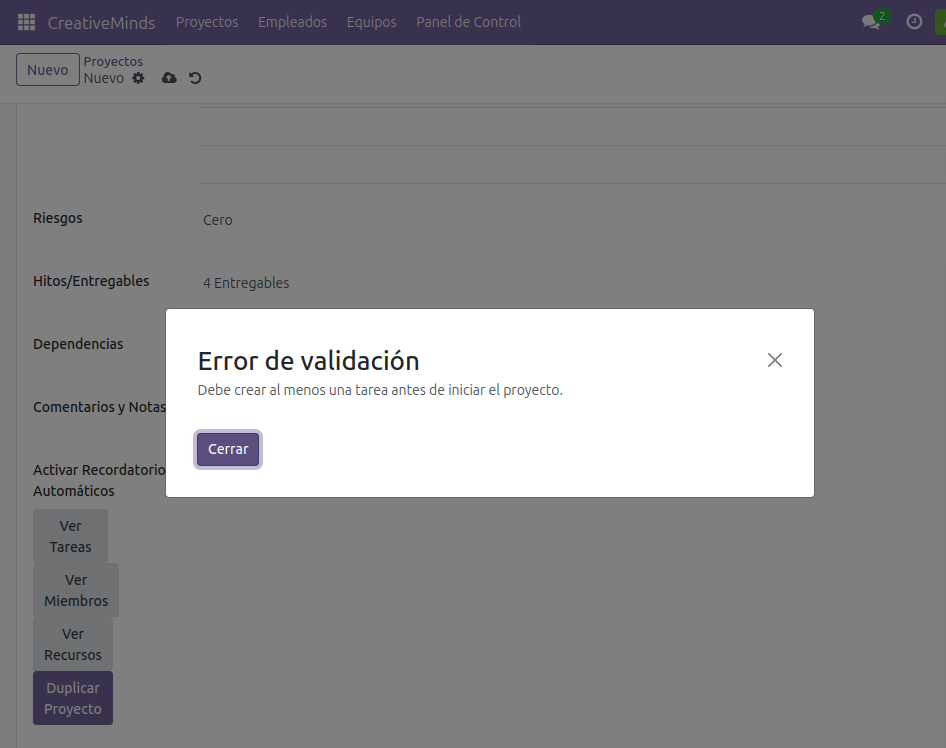
Caso: Intentamos crear una tarea sin nombre:



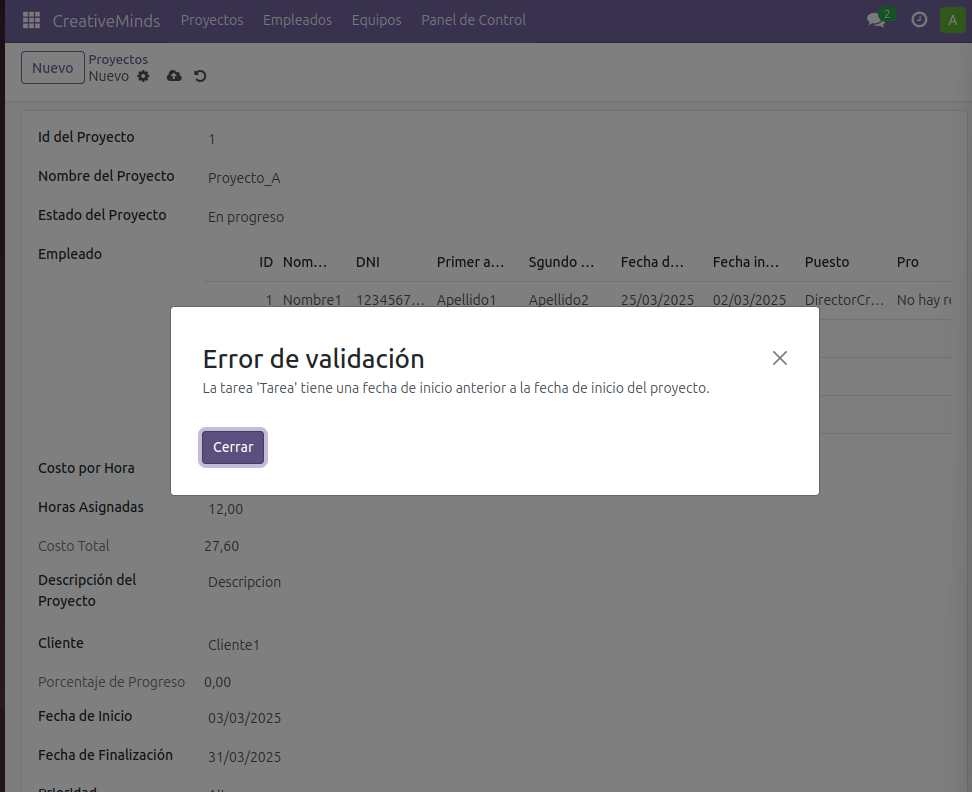


Una vez rellenado los datos, se ve así la tarea creada:

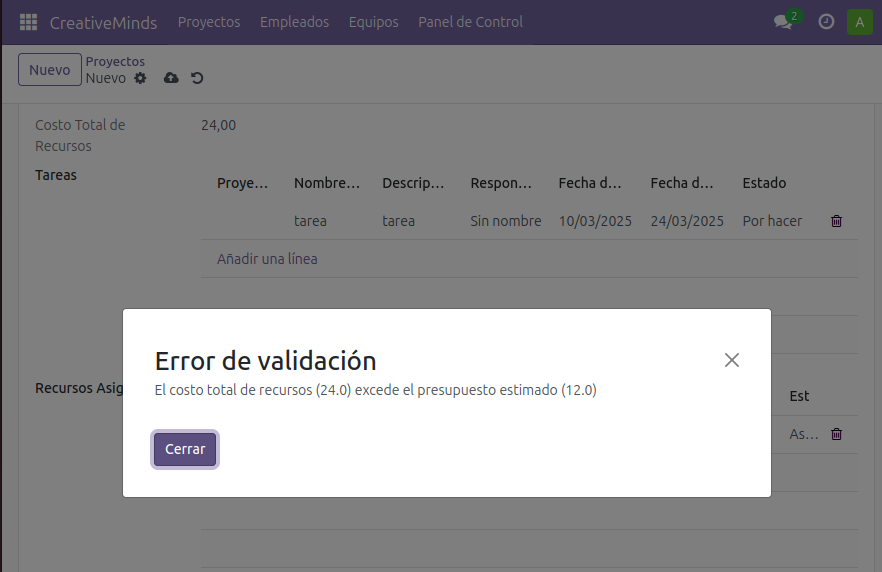
Pueba: Intentas ver los recursos y miembros sin haber inicializado tareas:



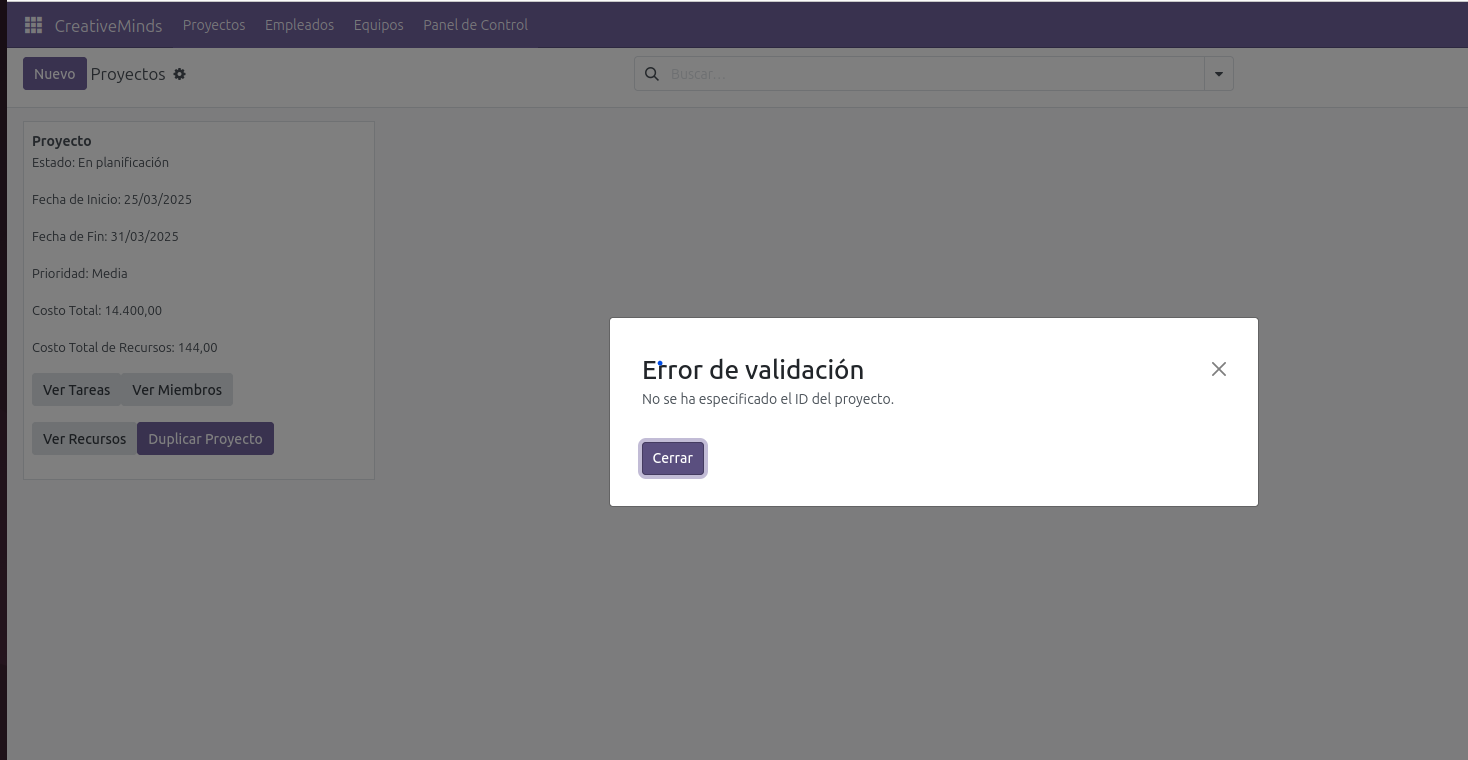
Prueba: Creas un proyecto con una tarea pero mal asignada la fecha:



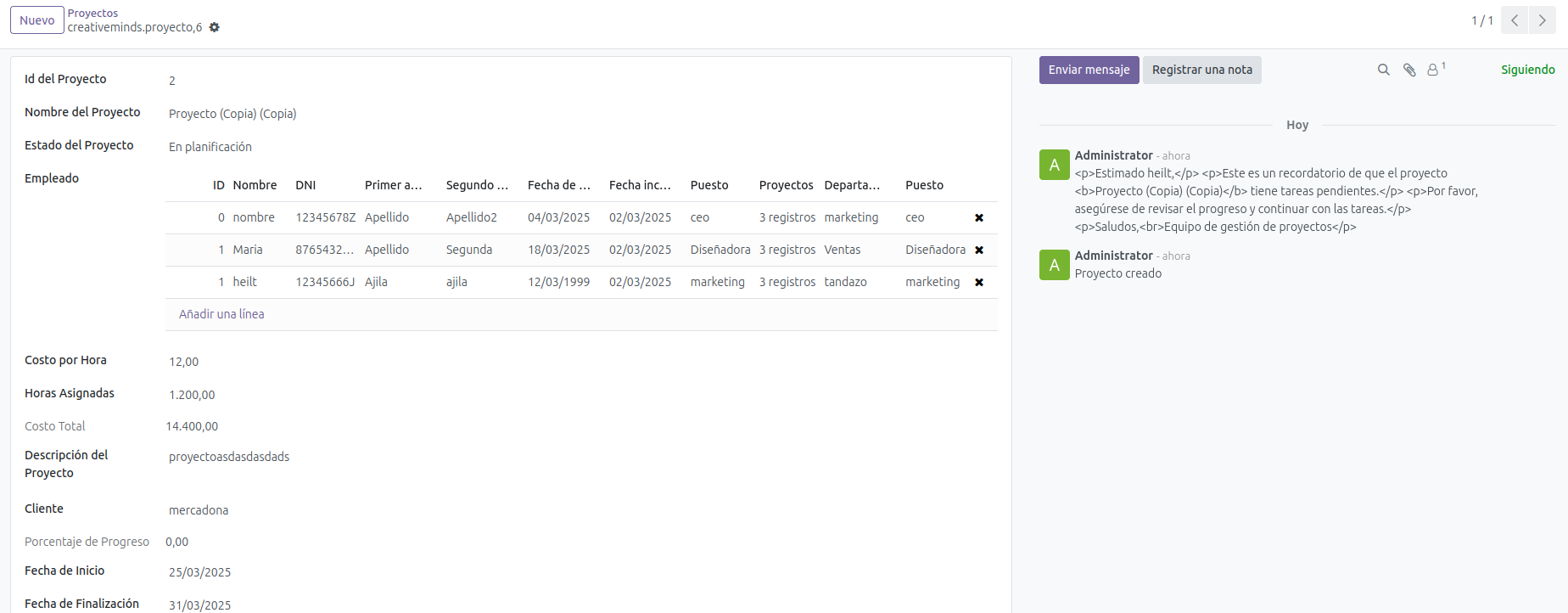
Prueba: Evitar que el usuario cree mal los recursos:

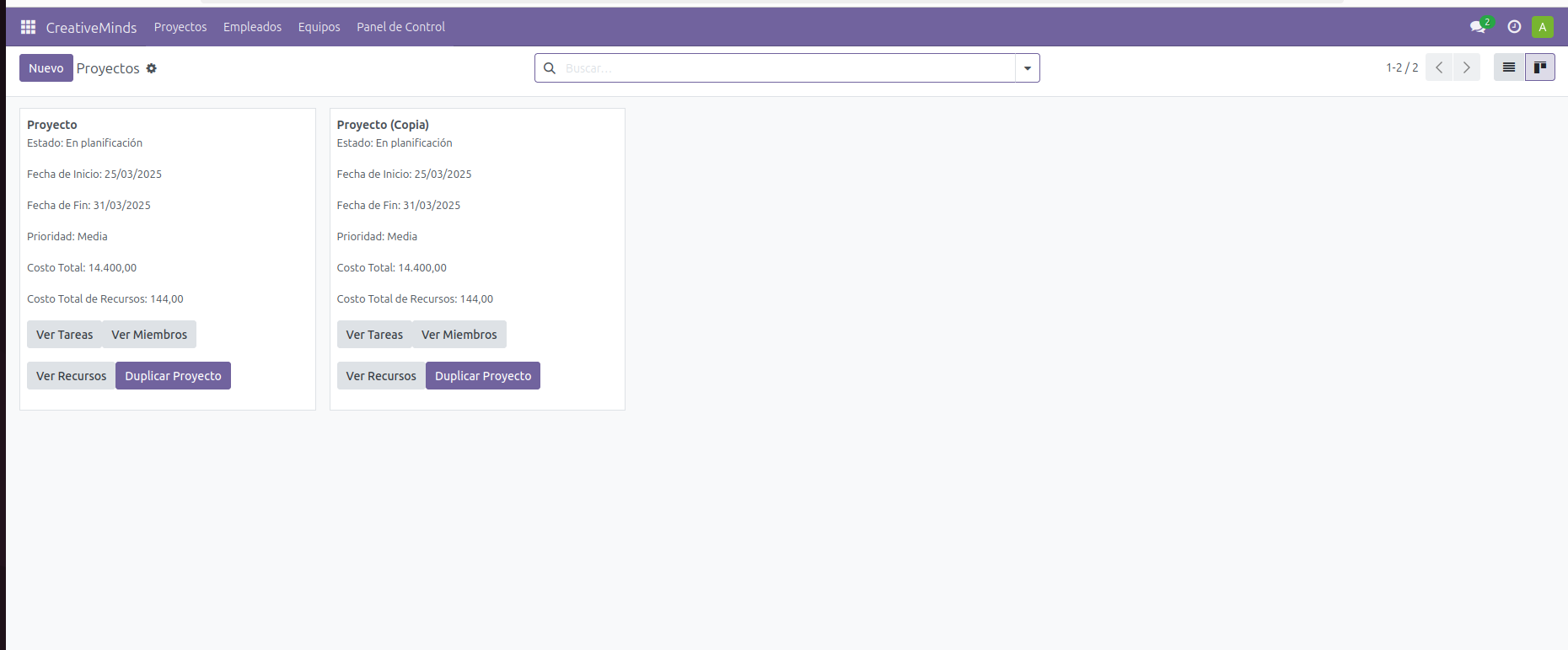


Caso: Intentas duplicar un proyecto sin especificar ID:



Caso: Duplicar el proyecto





**6. Conclusiones y Trabajo Futuro**

**6.1. Logros y limitaciones actuales**

|  |  |
| --- | --- |
| Logros | Limitaciones actuales |
| Almacenamiento estructurado de la información necesaria para la API. | Dificultad para ser manejado por una persona que no este familiarizada con el área de gestión de proyectos |
| Interconexión entre clases y ciertos modulos base como mail. | Falta de integración con otros módulos de Odoo como módulo Empleado |
| Creacion desde cero de las distintas clases. | La API actual requiere conocimientos técnicos específicos que podrían dificultar su implementación por equipos sin experiencia en desarrollo. |
|  | Nos falta añadir más funcionalidades y botones al módulo como ‘Eliminar copia’ desde la vista panorámica en Proyectos |

**6.2. Propuestas de mejora a futuro**

1. Uso de Odoo Studio porque asi nos facilitaría la personalización de la interfaz y la configuración de flujos de trabajo sin necesidad de modificar código, lo que agiliza la adaptación del módulo a necesidades específicas.

2. Mejor Seguridad y Control de Accesos tras la implementación de un registro de auditoría permitirá monitorear cambios en la base de datos, mejorando la trazabilidad y el cumplimiento normativo.

3. Integración con Inteligencia Artificial (IA), para que permita potenciar diversas áreas de empresas, como el área de CRM y Marketing haciendo que puedan realizar un análisis predictivo para segmentar clientes y automatizar campañas junto con los chatbots que proporcionarían un mejor soporte inteligente

4. Mejora en Rendimiento y Escalabilidad, tras la optimización de la gestión de bases de datos, mejorando las consultas para que soporten múltiples servidores para una mejor escalabilidad en la nube.

5. Realizar una integración avanzada con herramientas de Business Intelligence (BI) como Power BI. Además, potenciar el desarrollo de nuestra APIs para hacerla más avanzadas permitirá una conexión fluida con otros sistemas empresariales. Implementa endpoints de exportación de datos compatibles con formatos estándar (JSON, CSV, Excel) y desarrolla una capa de abstracción con especificaciones OpenAPI, facilitando la integración directa con herramientas BI como Power BI y sistemas empresariales externos.

*Llegamos a esta conclusión tras analizar la estructura actual de la API de Creative Minds, que ya maneja datos analíticos complejos pero carece de métodos estandarizados de exportación. Observamos que implementar endpoints específicos para exportación en formatos universales (JSON, CSV, Excel) y documentar la API con especificaciones OpenAPI no solo facilitaría la integración con herramientas de BI como Power BI, sino que también permitiría a otros sistemas empresariales consumir estos datos de forma automatizada, multiplicando así el valor de la información ya procesada por nuestra plataforma.*

**7. Referencias**

**7.1. Referencias y recursos externos utilizados**

[Configuración del sistema — documentación de Odoo - 17.0](https://www.odoo.com/documentation/17.0/es/administration/on_premise/deploy.html)

[SGE-UT5-Desarrollo de módulos Odoo16 - YouTube](https://www.youtube.com/playlist?list=PLOgymZ7xUgbb4iTyZxCjAPO_lExJCpSXr)

[Welcome to Flask — Flask Documentation (3.1.x)](https://flask.palletsprojects.com/en/stable/)

[SQLAlchemy Documentation — SQLAlchemy 2.0 Documentation](https://docs.sqlalchemy.org/en/20/)

[Pandas DataFrames](https://www.w3schools.com/python/pandas/pandas_dataframes.asp" \l ":~:text=Pandas%20DataFrames%20,table%20with%20rows%20and%20columns)

[How To Use a PostgreSQL Database in a Flask Application | DigitalOcean](https://www.digitalocean.com/community/tutorials/how-to-use-a-postgresql-database-in-a-flask-application)

[Tutorial Flask - Lección 5: Base de datos con Flask SQLAlchemy](https://j2logo.com/tutorial-flask-leccion-5-base-de-datos-con-flask-sqlalchemy/" \l ":~:text=app%20%3D%20Flask%28__name__%29%20app.config,SQLALCHEMY_TRACK_MODIFICATIONS%27%5D%20%3D%20False)

https://www.odoo.com/documentation/15.0/es/applications/studio/fields.html

**7.2. Código fuente comentado**

**creativeminds/\_\_manifest\_\_py:**

{

    # Nombre del módulo, se establece como 'CreativeMinds'

    'name': 'CreativeMinds',

    # Versión del módulo, se establece como '1.0'

    'version': '1.0',

    # Icono que se mostrará en el módulo, definido en una ruta específica

    'icon': '/creativeminds/static/description/icon.png',

    # Resumen del módulo, aquí se describe que es un módulo para la gestión de proyectos

    'summary': 'Modulo para la gestion de proyectos',

    # Descripción detallada del módulo, explicando su funcionalidad

    'description': """

        Gestion de proyectos

    """,

    # Autores del módulo, se especifican los nombres de las personas que lo desarrollaron

    'author': 'Heily Madelay Ajila Tandazo, Daniel Gonzalez Esteban',

    # Categoría del módulo, en este caso 'Marketing', lo que ayuda a clasificar el módulo en Odoo

    'category': 'Marketing',

    # Licencia bajo la que se distribuye el módulo, en este caso LGPL-3

    'license': 'LGPL-3',

    # Dependencias que necesita este módulo para funcionar, como 'base', 'hr', 'project', 'mail' y 'web'

    'depends': [

        'base',  # Dependencia básica de Odoo

        'hr',  # Módulo de recursos humanos

        'project',  # Módulo de gestión de proyectos

        'mail',  # Módulo para funcionalidades de correo

        'web',  # Módulo de vistas web

    ],

    # Archivos de datos que se cargarán al instalar el módulo, incluye acceso de seguridad, vistas y datos adicionales

    'data': [

        'security/ir.model.access.csv',  # Archivos de acceso para definir los permisos de los modelos

        'views/views.xml',  # Archivo de vistas que define la interfaz de usuario

        'data/dashboard\_data.xml',  # Datos relacionados con el panel de control

    ],

    # Se especifica que el módulo es instalable

    'installable': True,

    # Se establece que este módulo es una aplicación en Odoo

    'application': True,

    # Se marca como 'False' para que no se instale automáticamente

    'auto\_install': False,

}

**creativeminds/models/models.py:**

from odoo import models, fields, api  # Importa los módulos necesarios de Odoo para la creación de modelos y campos.

from odoo.exceptions import ValidationError  # Importa la excepción ValidationError para manejar errores de validación.

from datetime import date  # Importa el módulo date para trabajar con fechas.

import re  # Importa el módulo re para trabajar con expresiones regulares.

from dateutil.relativedelta import relativedelta  # Para realizar operaciones con fechas, como sumar o restar periodos.

from odoo.exceptions import UserError  # Para generar errores personalizados en Odoo.

import requests  # Para realizar solicitudes HTTP.

import json  # Para trabajar con datos en formato JSON.

class Proyecto(models.Model):

    \_name = 'creativeminds.proyecto'  # Nombre técnico del modelo en Odoo.

    \_description = 'Proyecto'  # Descripción del modelo.

    \_inherit = ['mail.thread', 'mail.activity.mixin']  # Herencia de funcionalidades de seguimiento de mensajes (para notificaciones y conversaciones).

    # Campos básicos del proyecto

    proyecto\_id = fields.Integer(string='Id del Proyecto', required=True)  # Identificador único del proyecto

    nombre = fields.Char(string='Nombre del Proyecto', required=True)  # Nombre del proyecto

    empleado\_id = fields.Many2many('creativeminds.empleado', string='Empleado')  # Empleados asignados al proyecto

    costo\_por\_hora = fields.Float(string='Costo por Hora')  # Costo por hora de trabajo en el proyecto

    horas\_asignadas = fields.Float(string='Horas Asignadas')  # Total de horas asignadas al proyecto

    costo\_total = fields.Float(string='Costo Total', compute='\_calcular\_costo\_total', store=True)  # Cálculo del costo total basado en las horas y costo por hora

    descripcion = fields.Text(string='Descripción del Proyecto')  # Descripción detallada del proyecto

    cliente = fields.Char(string='Cliente')  # Cliente asociado al proyecto

    @api.depends('costo\_por\_hora', 'horas\_asignadas')

    def \_calcular\_costo\_total(self):

        # Calcula el costo total del proyecto como el producto de costo por hora y horas asignadas

        self.costo\_total = self.costo\_por\_hora \* self.horas\_asignadas

    @api.constrains('costo\_por\_hora', 'horas\_asignadas')

    def \_verificar\_costo\_y\_horas(self):

        # Verifica que el costo por hora y las horas asignadas sean valores positivos

        if self.costo\_por\_hora < 0 or self.horas\_asignadas < 0:

            raise ValidationError("El costo por hora y las horas asignadas deben ser valores positivos.")

    # Campos relacionados con el estado y seguimiento del proyecto

    estado = fields.Selection([  # Estado del proyecto

        ('planificacion', 'En planificación'),

        ('en\_progreso', 'En progreso'),

        ('finalizado', 'Finalizado'),

        ('detenido', 'Detenido'),

    ], string='Estado del Proyecto', default='planificacion', tracking=True)

    porcentaje\_progreso = fields.Float(  # Porcentaje de progreso calculado

        string='Porcentaje de Progreso',

        compute='\_calcular\_progreso',

        store=True,

        tracking=True

    )

    # Fechas del proyecto

    fecha\_inicio = fields.Date(string='Fecha de Inicio')  # Fecha de inicio del proyecto

    fecha\_fin = fields.Date(string='Fecha de Finalización')  # Fecha de finalización del proyecto

    # Prioridad y responsables

    prioridad = fields.Selection([  # Nivel de prioridad del proyecto

        ('baja', 'Baja'),

        ('media', 'Media'),

        ('alta', 'Alta'),

    ], string='Prioridad', default='media')

    responsable\_id = fields.Many2one('creativeminds.empleado', string='Responsable')  # Responsable principal del proyecto

    # Presupuesto y recursos

    presupuesto\_estimado = fields.Float(string='Presupuesto Estimado')  # Presupuesto estimado del proyecto

    recursos\_ids = fields.One2many('creativeminds.recurso', 'proyecto\_id', string='Recursos Asignados')  # Recursos asignados al proyecto

    costo\_total\_recursos = fields.Float(string='Costo Total de Recursos',

                                     compute='\_calcular\_costo\_total\_recursos',

                                     store=True)  # Costo total de los recursos asignados

    @api.depends('recursos\_ids.costo\_total')

    def \_calcular\_costo\_total\_recursos(self):

        # Calcula el costo total de los recursos asignados al proyecto

        self.costo\_total\_recursos = sum(self.recursos\_ids.mapped('costo\_total'))

    @api.constrains('presupuesto\_estimado', 'costo\_total\_recursos')

    def \_verificar\_presupuesto(self):

        # Verifica que el costo total de los recursos no exceda el presupuesto estimado

        if self.costo\_total\_recursos > self.presupuesto\_estimado:

            raise ValidationError(

                f"El costo total de recursos ({self.costo\_total\_recursos}) "

                f"excede el presupuesto estimado ({self.presupuesto\_estimado})"

            )

    # Relaciones con tareas e indicadores

    tareas\_ids = fields.One2many('creativeminds.tarea', 'proyecto\_id', string='Tareas')  # Tareas asociadas al proyecto

    indicadores\_ids = fields.One2many('creativeminds.kpi', 'proyecto\_id', string='Indicadores de Desempeño')  # Indicadores de desempeño asociados al proyecto

    # Archivos y documentación

    imagen\_proyecto = fields.Binary(string='Imagen del Proyecto', attachment=True)  # Imagen del proyecto

    imagen\_filename = fields.Char(string='Nombre del archivo de imagen')  # Nombre del archivo de imagen

    documentacion\_tecnica = fields.Binary(string='Documentación Técnica', attachment=True)  # Documentación técnica del proyecto

    documentacion\_filename = fields.Char(string='Nombre del archivo de documentación')  # Nombre del archivo de documentación

    # Archivos adicionales

    archivos\_adicionales = fields.Many2many(

        'ir.attachment',

        'creativeminds\_attachment\_rel',

        'creativeminds\_id',

        'attachment\_id',

        string='Archivos Adicionales'

    )  # Archivos adicionales que se pueden asociar al proyecto

    # Campos de texto para detalles adicionales

    colaboradores = fields.Text(string='Agencias Colaboradoras')  # Agencias colaboradoras en el proyecto

    riesgos = fields.Text(string='Riesgos')  # Riesgos asociados al proyecto

    hitos = fields.Text(string='Hitos/Entregables')  # Hitos o entregables importantes del proyecto

    dependencias = fields.Text(string='Dependencias')  # Dependencias entre el proyecto y otros

    comentarios = fields.Text(string='Comentarios y Notas')  # Notas o comentarios sobre el proyecto

    # Configuración para recordatorios automáticos

    recordatorios\_automaticos = fields.Boolean(string='Activar Recordatorios Automáticos')

    # Verificación de fechas del proyecto

    @api.constrains('fecha\_inicio', 'fecha\_fin')

    def \_verificar\_fechas\_proyecto(self):

        # Asegura que la fecha de inicio no sea posterior a la fecha de finalización

        if self.fecha\_inicio and self.fecha\_fin and self.fecha\_inicio > self.fecha\_fin:

            raise ValidationError("La fecha de inicio no puede ser posterior a la fecha de finalización.")

    @api.constrains('presupuesto\_estimado')

    def \_verificar\_presupuesto\_estimado(self):

        # Verifica que el presupuesto estimado sea válido y suficiente para los recursos y tareas

        if self.presupuesto\_estimado <= 0:

            raise ValidationError("El presupuesto estimado debe ser mayor que cero.")

        num\_recursos = len(self.recursos\_ids)

        num\_tareas = len(self.tareas\_ids)

        presupuesto\_requerido = (num\_recursos \* 500) + (num\_tareas \* 200)  # Estimación de presupuesto requerido

        if (num\_recursos > 0 or num\_tareas > 0) and self.presupuesto\_estimado < presupuesto\_requerido:

            raise ValidationError(

                f"El presupuesto ({self.presupuesto\_estimado}) es insuficiente. "

                f"Se requieren al menos {round(presupuesto\_requerido, 2)} para cubrir recursos y tareas."

            )

    # Sobrescribimos el método 'create' para agregar lógica adicional al crear un proyecto.

    @api.model

    def create(self, valores):

        proyecto = super(Proyecto, self).create(valores) # Crear el proyecto como se haría normalmente.

        # Si los recordatorios automáticos están activados y el proyecto tiene un responsable asignado,

        # se crea una tarea inicial y se envía un recordatorio.

        if proyecto.recordatorios\_automaticos and proyecto.responsable\_id:

            self.env['creativeminds.tarea'].create({

                'nombre': f'Tarea inicial de proyecto: {proyecto.nombre}',# Nombre de la tarea inicial.

                'proyecto\_id': proyecto.id,  # Asociamos la tarea al proyecto.

                'responsable\_id': proyecto.responsable\_id.id,  # Asignamos al responsable del proyecto.

                'estado': 'pendiente',  # La tarea está pendiente al principio.

            })

            self.\_enviar\_recordatorio(proyecto)  # Enviamos un recordatorio al responsable.

        return proyecto  # Devolvemos el proyecto creado.

    # Método que actualiza el indicador de progreso del proyecto.

    def actualizar\_progreso\_indicador(self):

        # Buscamos el indicador específico relacionado con el progreso del proyecto.

        indicador = self.env['creativeminds.indicador'].search([

            ('proyecto\_id', '=', self.id),

            ('nombre', '=', 'Progreso del Proyecto')

        ], limit=1)

        if indicador:  # Si el indicador existe, actualizamos su valor.

            indicador.write({'valor': self.porcentaje\_progreso})

        return True

    # Método que envía una notificación al responsable del proyecto sobre el cambio de estado o progreso.

    def enviar\_notificacion\_proyecto(self, tipo\_notificacion='estado'):

        if not self.responsable\_id:  # Si no hay responsable, no se envía notificación.

            return

        # Determinamos el asunto de la notificación según el tipo.

        asunto = f"Cambio de Estado: Proyecto {self.nombre}" if tipo\_notificacion == 'estado' else f"Actualización de Progreso: Proyecto {self.nombre}"

        # Construimos el mensaje de notificación en formato HTML.

        mensaje = f"<p>Hola {self.responsable\_id.name},</p>"

        mensaje += f"<p>El estado del proyecto <b>{self.nombre}</b> ha cambiado a <b>{dict(self.\_fields['estado'].selection).get(self.estado)}</b>.</p>"

        mensaje += f"<p>Progreso actual: {round(self.porcentaje\_progreso, 2)}%</p>"

        # Enviamos el mensaje por correo utilizando el sistema de mensajes de Odoo.

        self.message\_post(

            body=mensaje,

            subject=asunto,

            partner\_ids=[self.responsable\_id.partner\_id.id]  # Enviamos el mensaje al partner del responsable.

        )

    # Método que envía un recordatorio al responsable del proyecto sobre las tareas pendientes.

    def enviar\_recordatorio(self, proyecto):

        if not proyecto.responsable\_id:  # Si no hay responsable asignado, no se envía el recordatorio.

            return

        # Construimos el mensaje de recordatorio en formato HTML.

        mensaje = f"""

            <p>Estimado {proyecto.responsable\_id.name},</p>

            <p>Este es un recordatorio de que el proyecto <b>{proyecto.nombre}</b> tiene tareas pendientes.</p>

            <p>Por favor, asegúrese de revisar el progreso y continuar con las tareas.</p>

            <p>Saludos,<br>Equipo de gestión de proyectos</p>

        """

        # Enviamos el recordatorio por mensaje en Odoo.

        proyecto.message\_post(

            body=mensaje,

            subject=f"Recordatorio: Proyecto {proyecto.nombre} - Tareas pendientes",

            partner\_ids=[proyecto.responsable\_id.partner\_id.id]  # Enviamos el mensaje al partner del responsable.

        )

        # Creamos una actividad para el responsable para asegurarnos de que se realice el seguimiento de las tareas pendientes.

        actividad\_tipo = self.env.ref('mail.mail\_activity\_data\_todo')

        modelo\_id = self.env['ir.model'].\_get\_id('creativeminds.proyecto')

        self.env['mail.activity'].create({

            'activity\_type\_id': actividad\_tipo.id,

            'res\_model\_id': modelo\_id,

            'res\_id': proyecto.id,

            'user\_id': proyecto.responsable\_id.id,

            'summary': f"Recordatorio: {proyecto.nombre} - Tareas pendientes",

            'note': f"Este es un recordatorio para que revises las tareas pendientes del proyecto {proyecto.nombre}.",

        })

    # Método que calcula el porcentaje de progreso del proyecto basado en las tareas completadas.

    @api.depends('tareas\_ids.estado')

    def \_calcular\_progreso(self):

        total\_tareas = len(self.tareas\_ids)  # Contamos el total de tareas.

        tareas\_completadas = len(self.tareas\_ids.filtered(lambda t: t.estado == 'completada'))

        # Calculamos el progreso como el porcentaje de tareas completadas.

        self.porcentaje\_progreso = (tareas\_completadas / total\_tareas \* 100) if total\_tareas > 0 else 0.0

    # Método que obtiene un resumen detallado del proyecto.

    def obtener\_resumen\_proyecto(self):

        self.ensure\_one()  # Aseguramos que solo haya un registro.

        return {

            'nombre': self.nombre,

            'estado': self.estado,

            'progreso': self.porcentaje\_progreso,

            'presupuesto': {

                'estimado': self.presupuesto\_estimado,

                'actual': self.costo\_total\_recursos,

                'disponible': self.presupuesto\_estimado - self.costo\_total\_recursos

            },

            'tareas': {

                'total': len(self.tareas\_ids),

                'completadas': len(self.tareas\_ids.filtered(lambda t: t.estado == 'completada')),

                'en\_progreso': len(self.tareas\_ids.filtered(lambda t: t.estado == 'en\_progreso')),

                'pendientes': len(self.tareas\_ids.filtered(lambda t: t.estado == 'pendiente'))

            }

        }

    def duplicar\_proyecto(self):

        """

        Función para duplicar un proyecto existente.

        No requiere argumentos adicionales ya que opera sobre el registro actual (self).

        """

        self.ensure\_one()

        # Crear una copia del proyecto actual

        valores = {

            'nombre': self.nombre + ' (Copia)',

            'estado': 'planificacion',

            'proyecto\_id': self.proyecto\_id + 1,  # Incrementamos el ID para la copia

            'empleado\_id': [(6, 0, self.empleado\_id.ids)],  # Preservar relaciones many2many

            'costo\_por\_hora': self.costo\_por\_hora,

            'horas\_asignadas': self.horas\_asignadas,

            'descripcion': self.descripcion,

            'cliente': self.cliente,

            'fecha\_inicio': self.fecha\_inicio,

            'fecha\_fin': self.fecha\_fin,

            'prioridad': self.prioridad,

            'responsable\_id': self.responsable\_id.id if self.responsable\_id else False,

            'presupuesto\_estimado': self.presupuesto\_estimado,

            'riesgos': self.riesgos,

            'hitos': self.hitos,

            'dependencias': self.dependencias,

            'comentarios': self.comentarios,

            'recordatorios\_automaticos': self.recordatorios\_automaticos,

        }

        # Crear nuevo proyecto con los valores copiados

        nuevo\_proyecto = self.create(valores)

        # Duplicar las tareas asociadas

        for tarea in self.tareas\_ids:

            tarea\_valores = {

                'proyecto\_id': nuevo\_proyecto.id,

                'nombre': tarea.nombre,

                'descripcion': tarea.descripcion,

                'responsable\_id': tarea.responsable\_id.id if tarea.responsable\_id else False,

                'fecha\_inicio': tarea.fecha\_inicio,

                'fecha\_fin': tarea.fecha\_fin,

                'estado': 'pendiente',  # Las tareas duplicadas comienzan como pendientes

            }

            self.env['creativeminds.tarea'].create(tarea\_valores)

        # Duplicar los recursos asignados

        for recurso in self.recursos\_ids:

            recurso\_valores = {

                'proyecto\_id': nuevo\_proyecto.id,

                'nombre': recurso.nombre,

                'empleado\_id': [(6, 0, recurso.empleado\_id.ids)] if recurso.empleado\_id else [],

                'costo\_por\_hora': recurso.costo\_por\_hora,

                'horas\_asignadas': recurso.horas\_asignadas,

                'fecha\_inicio': recurso.fecha\_inicio,

                'fecha\_fin': recurso.fecha\_fin,

                'estado': 'borrador',  # Los recursos duplicados comienzan como borrador

            }

            self.env['creativeminds.recurso'].create(recurso\_valores)

        # Mostrar el formulario del nuevo proyecto

        return {

            'name': 'Proyecto Duplicado',

            'type': 'ir.actions.act\_window',

            'res\_model': 'creativeminds.proyecto',

            'view\_mode': 'form',

            'res\_id': nuevo\_proyecto.id,

            'target': 'current',

        }

    # Método para verificar que la descripción, cliente y responsable sean válidos antes de hacer cambios.

    @api.constrains('descripcion', 'cliente', 'responsable\_id')

    def \_verificar\_campos\_importantes(self):

        if self.descripcion and len(self.descripcion.strip()) < 10:  # La descripción debe tener al menos 10 caracteres.

            raise ValidationError("La descripción del proyecto debe tener al menos 10 caracteres.")

        if self.estado in ['en\_progreso', 'finalizado'] and not self.cliente:  # El cliente es obligatorio si el estado es 'en progreso' o 'finalizado'.

            raise ValidationError("Debe especificar un cliente antes de cambiar el proyecto a 'En progreso' o 'Finalizado'.")

        if self.estado != 'planificacion' and not self.responsable\_id:  # El responsable es obligatorio si el proyecto no está en planificación.

            raise ValidationError("Debe asignar un responsable antes de avanzar con el proyecto.")

    # Método para verificar que los campos de planificación son correctos para proyectos de alta prioridad.

    @api.constrains('riesgos', 'hitos')

    def \_verificar\_campos\_planificacion(self):

        if self.prioridad == 'alta':  # Si el proyecto es de alta prioridad.

            if not self.riesgos:  # Los riesgos deben estar definidos.

                raise ValidationError("Para proyectos de alta prioridad, es obligatorio definir los riesgos.")

            if not self.hitos:  # Los hitos deben estar definidos.

                raise ValidationError("Para proyectos de alta prioridad, es obligatorio definir los hitos/entregables.")

    # Método para verificar que haya al menos un recurso asignado antes de cambiar el estado a 'en progreso'.

    @api.constrains('recursos\_ids')

    def \_verificar\_recursos\_minimos(self):

        if self.estado != 'planificacion' and not self.recursos\_ids:  # Si el proyecto no está en planificación y no tiene recursos asignados.

            raise ValidationError("Debe asignar al menos un recurso antes de iniciar el proyecto.")

    # Método para verificar que las fechas de inicio y fin, así como las tareas, sean coherentes.

    @api.constrains('fecha\_inicio', 'fecha\_fin', 'estado', 'tareas\_ids')

    def \_verificar\_fechas\_y\_tareas(self):

        if self.estado == 'en\_progreso':  # Si el proyecto está en progreso.

            if not self.fecha\_inicio:  # La fecha de inicio debe estar definida.

                raise ValidationError("Debe establecer una fecha de inicio antes de comenzar el proyecto.")

            if not self.fecha\_fin:  # La fecha de fin debe estar definida.

                raise ValidationError("Debe establecer una fecha de finalización antes de comenzar el proyecto.")

            if not self.tareas\_ids:  # Deben existir al menos una tarea asociada.

                raise ValidationError("Debe crear al menos una tarea antes de iniciar el proyecto.")

            for tarea in self.tareas\_ids:

                if tarea.fecha\_inicio and tarea.fecha\_inicio < self.fecha\_inicio:  # La tarea no puede empezar antes de la fecha de inicio del proyecto.

                    raise ValidationError(f"La tarea '{tarea.nombre}' tiene una fecha de inicio anterior a la fecha de inicio del proyecto.")

                if tarea.fecha\_fin and tarea.fecha\_fin > self.fecha\_fin:  # La tarea no puede terminar después de la fecha de finalización del proyecto.

                    raise ValidationError(f"La tarea '{tarea.nombre}' tiene una fecha de finalización posterior a la fecha de fin del proyecto.")

    # Función para ver las tareas del proyecto

    def ver\_tareas(self):

        return {

            'name': 'Tareas del Proyecto',

            'type': 'ir.actions.act\_window',

            'res\_model': 'creativeminds.tarea',

            'view\_mode': 'tree,form',

            'domain': [('proyecto\_id', '=', self.id)],

            'context': {'default\_proyecto\_id': self.id},

            'target': 'current',

        }

    # Función para ver los recursos del proyecto

    def ver\_recursos(self):

        return {

            'name': 'Recursos del Proyecto',

            'type': 'ir.actions.act\_window',

            'res\_model': 'creativeminds.recurso',

            'view\_mode': 'tree,form',

            'domain': [('proyecto\_id', '=', self.id)],

            'context': {'default\_proyecto\_id': self.id},

            'target': 'current',

        }

    # Función para ver los miembros del proyecto

    def ver\_miembros(self):

        return {

            'name': 'Miembros del Proyecto',

            'type': 'ir.actions.act\_window',

            'res\_model': 'creativeminds.empleado',

            'view\_mode': 'tree,form',

            'domain': [('proyecto\_id', '=', self.id)],

            'context': {'default\_proyecto\_id': self.id},

            'target': 'current',

        }

class Recurso(models.Model):

    \_name = 'creativeminds.recurso'

    \_description = 'Recursos del Proyecto'

    # Campos básicos

    nombre = fields.Char(string='Nombre del Recurso', required=True)  # Nombre del recurso (obligatorio).

    empleado\_id = fields.Many2many('creativeminds.empleado', string='Empleado')  # Relación con los empleados asignados al recurso.

    proyecto\_id = fields.Many2one('creativeminds.proyecto', string='Proyecto')  # Relación con el proyecto al que pertenece el recurso.

    # Costos y presupuesto

    costo\_por\_hora = fields.Float(string='Costo por Hora')  # Costo por hora del recurso.

    horas\_asignadas = fields.Float(string='Horas Asignadas')  # Número de horas asignadas al recurso.

    costo\_total = fields.Float(string='Costo Total', compute='\_compute\_costo\_total', store=True)  # Cálculo automático del costo total basado en el costo por hora y las horas asignadas.

    @api.depends('costo\_por\_hora', 'horas\_asignadas')  # Cuando cambian el costo o las horas, se recalcula el costo total.

    def \_compute\_costo\_total(self):

        for record in self:

            record.costo\_total = record.costo\_por\_hora \* record.horas\_asignadas  # El costo total es el costo por hora multiplicado por las horas asignadas.

    # Fechas de asignación

    fecha\_inicio = fields.Date(string='Fecha de Inicio')  # Fecha de inicio de la asignación del recurso.

    fecha\_fin = fields.Date(string='Fecha de Fin')  # Fecha de finalización de la asignación del recurso.

    # Estado del recurso

    estado = fields.Selection([  # Definimos los posibles estados del recurso.

        ('borrador', 'Borrador'),  # Estado inicial.

        ('asignado', 'Asignado'),  # Cuando el recurso ya está asignado al proyecto.

        ('en\_progreso', 'En Progreso'),  # Cuando el recurso está trabajando activamente en el proyecto.

        ('completado', 'Completado')  # Cuando el recurso ha finalizado su tarea.

    ], string='Estado', default='borrador')  # El estado por defecto es 'borrador'.

class Tarea(models.Model):

    \_name = 'creativeminds.tarea'

    \_description = 'Tareas del Proyecto'

    # Campos básicos

    proyecto\_id = fields.Many2one('creativeminds.proyecto', string='Proyecto')  # Relación con el proyecto al que pertenece la tarea.

    nombre = fields.Char(string='Nombre de la Tarea', required=True)  # Nombre de la tarea (obligatorio).

    descripcion = fields.Text(string='Descripción')  # Descripción opcional de la tarea.

    responsable\_id = fields.Many2one('creativeminds.empleado', string='Responsable')  # Relación con el empleado que es responsable de la tarea.

    fecha\_inicio = fields.Date(string='Fecha de Inicio')  # Fecha en la que la tarea debería comenzar.

    fecha\_fin = fields.Date(string='Fecha de Finalización')  # Fecha en la que la tarea debe finalizar.

    tareas\_ids = fields.One2many('creativeminds.tarea', string='Tareas')  # Tareas asociadas al empleado

    estado = fields.Selection([  # Selección de estados de la tarea.

        ('pendiente', 'Por hacer'),  # Estado cuando la tarea aún no se ha comenzado.

        ('en\_progreso', 'En progreso'),  # Estado cuando la tarea está siendo trabajada.

        ('completada', 'Completada'),  # Estado cuando la tarea ha sido finalizada.

    ], string='Estado', default='pendiente')  # El estado inicial es "pendiente" por defecto.

    # Método de validación de fechas

    @api.constrains('fecha\_inicio', 'fecha\_fin')  # Este decorador valida las fechas de inicio y fin.

    def \_verificar\_fechas\_tarea(self):

        if self.fecha\_inicio and self.fecha\_fin and self.fecha\_inicio > self.fecha\_fin:  # Si la fecha de inicio es mayor que la de fin, lanza un error.

            raise ValidationError("La fecha de inicio no puede ser posterior a la fecha de finalización.")  # Lanza un error de validación si las fechas no son correctas.

#Indicadores de Desempeño

class KPI(models.Model):

    \_name = 'creativeminds.kpi'

    \_description = 'Indicadores Clave de Rendimiento'

    # Campos de datos

    proyecto\_id = fields.Many2one('creativeminds.proyecto', string='Proyecto')  # Relación con el proyecto al que pertenece el KPI.

    nombre = fields.Char(string='Nombre del KPI', required=True)  # Nombre del KPI, que es obligatorio.

    valor = fields.Float(string='Valor')  # Valor actual del KPI, que puede ser un número decimal.

    objetivo = fields.Float(string='Objetivo')  # Objetivo o meta del KPI, también como número decimal.

class Empleado(models.Model):

    \_name = 'creativeminds.empleado'

    \_description = 'Empleados del Proyecto'

    \_inherit = ['mail.thread']

    empleado\_id = fields.Integer(string='ID', required=True)  # Campo entero para almacenar el ID del empleado (requerido).

    partner\_id = fields.Many2one('res.partner', string='Contacto Asociado')  # Relación Many2one con el modelo 'res.partner' para asociar un contacto.

    name = fields.Char(string='Nombre', required=True)  # Campo de texto para almacenar el nombre del empleado (requerido).

    dni = fields.Char(string='DNI', size=9, required=True)  # Campo de texto para almacenar el DNI del empleado (requerido, con un tamaño máximo de 9 caracteres).

    apellido1 = fields.Char(string='Primer apellido')  # Campo de texto para almacenar el primer apellido del empleado.

    apellido2 = fields.Char(string='Segundo apellido')  # Campo de texto para almacenar el segundo apellido del empleado.

    fecha\_nacimiento = fields.Date(string='Fecha de nacimiento')  # Campo de fecha para almacenar la fecha de nacimiento del empleado.

    fecha\_incorporacion = fields.Date(string='Fecha incorporacion', default=lambda self: fields.Datetime.now(), readonly=True)  # Fecha de incorporación con valor por defecto de la fecha y hora actual, solo lectura.

    foto = fields.Image(string='Foto', max\_width=200, max\_height=200)  # Campo de imagen para almacenar la foto del empleado con un tamaño máximo de 200x200 píxeles.

    proyecto\_id = fields.Many2many('creativeminds.proyecto', string='Proyectos')  # Relación Many2many con el modelo 'creativeminds.proyecto' para asociar proyectos al empleado.

    departamento = fields.Char(string='Departamento')  # Campo de texto para almacenar el departamento del empleado.

    puesto = fields.Char(string='Puesto')  # Campo de texto para almacenar el puesto del empleado.

    equipo\_id = fields.Many2many('creativeminds.equipo', string='Equipos')  # Relación Many2many con el modelo 'creativeminds.equipo' para asociar equipos al empleado.

    # Estado de disponibilidad

    disponibilidad = fields.Selection([  # Campo para gestionar la disponibilidad del empleado.

        ('disponible', 'Disponible'),  # El empleado está disponible para trabajar.

        ('asignado', 'Asignado'),  # El empleado está asignado a un proyecto.

        ('parcial', 'Parcialmente Disponible'),  # El empleado está parcialmente disponible.

        ('no\_disponible', 'No Disponible')  # El empleado no está disponible.

    ], string='Disponibilidad', default='disponible')  # Valor por defecto es "disponible".

    # Restricción en el campo DNI: formato válido

    @api.constrains('dni')

    def \_check\_dni(self):

        regex = re.compile(r'[0-9]{8}[A-Z]\Z', re.I)  # Expresión regular para verificar el formato correcto del DNI (8 dígitos seguidos de una letra).

        for record in self:

            if not regex.match(record.dni):  # Si el formato no es válido, lanza una excepción.

                raise ValidationError('ERROR. Formato DNI incorrecto.')

    @api.constrains('fecha\_nacimiento')

    def \_check\_edad\_minima(self):

        for record in self:

            if record.fecha\_nacimiento:

                # Calcular la edad actual basada en la fecha de nacimiento

                edad = relativedelta(date.today(), record.fecha\_nacimiento).years

                if edad < 16:

                    raise ValidationError("El empleado debe tener al menos 16 años para poder trabajar.")

                if record.fecha\_nacimiento > date.today():

                    raise ValidationError("La fecha de nacimiento no puede estar en el futuro.")

    # Restricción SQL: asegura que el DNI sea único en la base de datos.

    \_sql\_constraints = [

        ('DNI\_unico', 'UNIQUE(dni)', "El DNI debe ser único")  # Restricción de unicidad en el campo DNI.

    ]

    @api.model

    def create(self, vals):

        record = super(Empleado, self).create(vals)

        # Enviar una notificación al crear el empleado

        record.message\_post(

            body=f"Se ha creado un nuevo empleado: {record.name}.",

            subject="Nuevo Empleado",

            partner\_ids=[record.partner\_id.id]  # Enviamos el mensaje al partner (empleado) creado

        )

        return record

class Equipo(models.Model):

    \_name = 'creativeminds.equipo'

    \_description = 'Equipos de Trabajo'

    # Campos básicos de información del equipo

    equipo\_id = fields.Integer(string='ID Grupo', required=True)  # ID único del equipo, obligatorio.

    nombre = fields.Char(string='Nombre', required=True)  # Nombre del equipo, obligatorio.

    empleado\_id = fields.Many2many('creativeminds.empleado', string='Empleado')  # Relación de muchos a muchos con los empleados del equipo.

    responsable\_id = fields.Many2one('creativeminds.empleado', string='Responsable')  # Relación con un solo responsable del equipo.

    descripcion = fields.Text(string='Descripcion del equipo')  # Descripción del equipo, opcional.

    # Campo calculado: número de miembros en el equipo

    n\_miembros = fields.Integer(string='Número de Miembros', compute='\_compute\_n\_miembros')  # Número de miembros calculado dinámicamente.

    # Método para calcular el número de miembros en el equipo

    def \_compute\_n\_miembros(self):

        for equipo in self:

            equipo.n\_miembros = len(equipo.empleado\_id)  # Asigna el número de miembros en función de la cantidad de empleados asociados al equipo.

class DashboardMetricas(models.Model):

    \_name = 'creativeminds.metrica'

    \_description = 'Métricas del Dashboard'

    panel\_id = fields.Many2one('creativeminds.control.panel', string='Panel de Control')

    fecha\_actualizacion = fields.Datetime(string='Fecha de Actualización')

    # Métricas generales

    total\_proyectos = fields.Integer(string='Total de Proyectos')

    proyectos\_en\_progreso = fields.Integer(string='Proyectos en Progreso')

    proyectos\_finalizados = fields.Integer(string='Proyectos Finalizados')

    proyectos\_retrasados = fields.Integer(string='Proyectos Retrasados')

    progreso\_promedio = fields.Float(string='Progreso Promedio (%)')

    presupuesto\_total = fields.Float(string='Presupuesto Total')

    costo\_actual\_total = fields.Float(string='Costo Actual Total')

    eficiencia\_presupuestaria = fields.Float(string='Eficiencia Presupuestaria (%)')

    total\_tareas = fields.Integer(string='Total de Tareas')

    tareas\_completadas = fields.Integer(string='Tareas Completadas')

    tareas\_pendientes = fields.Integer(string='Tareas Pendientes')

    empleados\_disponibles = fields.Integer(string='Empleados Disponibles')

class Recomendaciones(models.Model):

    \_name = 'creativeminds.recomendacion'

    \_description = 'Recomendaciones del Sistema'

    \_order = 'prioridad asc'

    panel\_id = fields.Many2one('creativeminds.control.panel', string='Panel de Control')

    descripcion = fields.Text(string='Descripción')

    prioridad = fields.Integer(string='Prioridad')

    fecha = fields.Date(string='Fecha')

    estado = fields.Selection([

        ('pendiente', 'Pendiente'),

        ('en\_proceso', 'En Proceso'),

        ('implementada', 'Implementada'),

        ('descartada', 'Descartada')

    ], string='Estado', default='pendiente')

    notas = fields.Text(string='Notas')

class PanelDeControl(models.Model):

    \_name = 'creativeminds.control.panel'  # Nombre técnico del modelo en Odoo.

    \_description = 'Panel de Control'  # Descripción del modelo: se usa para gestionar un panel de control de proyectos.

    nombre = fields.Char(string='Nombre', required=True)

    proyectos\_ids = fields.Many2many('creativeminds.proyecto', string='Proyectos en el Panel')

    configuracion = fields.Text(string='Configuración del Panel')

    # Campos para el análisis FODA

    fortalezas = fields.Text(string='Fortalezas')

    debilidades = fields.Text(string='Debilidades')

    oportunidades = fields.Text(string='Oportunidades')

    amenazas = fields.Text(string='Amenazas')

    # Campo para almacenar la fecha de la última actualización

    ultima\_actualizacion = fields.Datetime(string='Última Actualización')

    @api.model

    def \_ensure\_default\_record(self):

        """Asegura que exista al menos un registro en el modelo para mostrar el dashboard."""

        if not self.search([], limit=1):

            self.create({'nombre': 'Panel Global'})

        return True

    @api.model

    def \_init\_dashboard(self):

        self.\_ensure\_default\_record()

        return True

    @api.model

    def action\_load\_data\_from\_api(self):

        """Carga datos desde la API externa y actualiza registros en Odoo."""

        url = "http://127.0.0.1:5000/api/dashboard"  # Endpoint correcto

        try:

            response = requests.get(url, timeout=10)

            if response.status\_code == 200:

                data = response.json()

                panel\_default = self.search([], limit=1)

                if not panel\_default:

                    panel\_default = self.create({'nombre': 'Panel Global'})

                # 1. Procesar métricas generales

                if 'metricas' in data and panel\_default:

                    metricas = data['metricas']

                    # Crear o actualizar un registro de métricas

                    # Primero, buscar si ya existe un registro de métricas para este panel

                    metrica\_existente = self.env['creativeminds.metrica'].search([

                        ('panel\_id', '=', panel\_default.id)

                    ], limit=1)

                    metrica\_values = {

                        'total\_proyectos': metricas.get('total\_proyectos', 0),

                        'proyectos\_en\_progreso': metricas.get('proyectos\_en\_progreso', 0),

                        'proyectos\_finalizados': metricas.get('proyectos\_finalizados', 0),

                        'proyectos\_retrasados': metricas.get('proyectos\_retrasados', 0),

                        'progreso\_promedio': metricas.get('progreso\_promedio', 0),

                        'presupuesto\_total': metricas.get('presupuesto\_total', 0),

                        'costo\_actual\_total': metricas.get('costo\_actual\_total', 0),

                        'eficiencia\_presupuestaria': metricas.get('eficiencia\_presupuestaria', 0),

                        'total\_tareas': metricas.get('total\_tareas', 0),

                        'tareas\_completadas': metricas.get('tareas\_completadas', 0),

                        'tareas\_pendientes': metricas.get('tareas\_pendientes', 0),

                        'empleados\_disponibles': metricas.get('empleados\_disponibles', 0),

                        'fecha\_actualizacion': fields.Datetime.now(),

                        'panel\_id': panel\_default.id

                    }

                    if metrica\_existente:

                        metrica\_existente.write(metrica\_values)

                    else:

                        self.env['creativeminds.metrica'].create(metrica\_values)

                # 2. Procesar proyectos destacados

                if 'proyectos\_destacados' in data:

                    # Obtener los IDs de proyectos destacados para actualizar el panel

                    proyectos\_destacados\_ids = []

                    for proyecto\_data in data['proyectos\_destacados']:

                        # Buscar si el proyecto ya existe

                        proyecto\_existente = self.env['creativeminds.proyecto'].search([

                            ('proyecto\_id', '=', proyecto\_data.get('proyecto\_id'))

                        ], limit=1)

                        proyecto\_values = {

                            'nombre': proyecto\_data.get('nombre', ''),

                            'estado': proyecto\_data.get('estado', 'planificacion'),

                            'fecha\_inicio': proyecto\_data.get('fecha\_inicio', False),

                            'fecha\_fin': proyecto\_data.get('fecha\_fin', False),

                            'presupuesto\_estimado': proyecto\_data.get('presupuesto\_estimado', 0),

                            'costo\_total\_recursos': proyecto\_data.get('costo\_total\_recursos', 0),

                            'porcentaje\_progreso': proyecto\_data.get('porcentaje\_progreso', 0)

                        }

                        if proyecto\_existente:

                            proyecto\_existente.write(proyecto\_values)

                            proyectos\_destacados\_ids.append(proyecto\_existente.id)

                        else:

                            proyecto\_values['proyecto\_id'] = proyecto\_data.get('proyecto\_id')

                            nuevo\_proyecto = self.env['creativeminds.proyecto'].create(proyecto\_values)

                            proyectos\_destacados\_ids.append(nuevo\_proyecto.id)

                    # Actualizar la relación many2many del panel con los proyectos destacados

                    if panel\_default and proyectos\_destacados\_ids:

                        panel\_default.write({

                            'proyectos\_ids': [(6, 0, proyectos\_destacados\_ids)]  # Reemplaza todos los proyectos con los nuevos

                        })

                # 3. Procesar y guardar el análisis FODA (como ya te mostré antes)

                if 'analisis' in data and panel\_default:

                    analisis = data['analisis']

                    # Convertir listas a texto con formato

                    fortalezas = '\n'.join(['• ' + f for f in analisis.get('fortalezas', [])])

                    debilidades = '\n'.join(['• ' + d for d in analisis.get('debilidades', [])])

                    oportunidades = '\n'.join(['• ' + o for o in analisis.get('oportunidades', [])])

                    amenazas = '\n'.join(['• ' + a for a in analisis.get('amenazas', [])])

                    panel\_default.write({

                        'fortalezas': fortalezas,

                        'debilidades': debilidades,

                        'oportunidades': oportunidades,

                        'amenazas': amenazas,

                        'ultima\_actualizacion': fields.Datetime.now()

                    })

                # 4. Procesar recomendaciones (opcional)

                if 'recomendaciones' in data and panel\_default:

                    recomendaciones = data['recomendaciones']

                    # Eliminar recomendaciones anteriores

                    self.env['creativeminds.recomendacion'].search([

                        ('panel\_id', '=', panel\_default.id)

                    ]).unlink()

                    # Crear nuevas recomendaciones

                    for i, recomendacion\_texto in enumerate(recomendaciones):

                        self.env['creativeminds.recomendacion'].create({

                            'panel\_id': panel\_default.id,

                            'descripcion': recomendacion\_texto,

                            'prioridad': i + 1,  # Prioridad basada en el orden

                            'fecha': fields.Date.today()

                        })

                return {

                    'status': 'ok',

                    'message': 'Datos del dashboard cargados correctamente'

                }

            else:

                return {

                    'status': 'error',

                    'message': f"Error {response.status\_code}: {response.text}"

                }

        except requests.exceptions.RequestException as e:

            raise UserError(f"Error al conectar con la API: {str(e)}")

**creativeminds/views/views.xml:**

<?xml version="1.0" encoding="UTF-8"?>

<odoo>

    <!-- Views -->

    <!-- Definición de vistas para la gestión de proyectos -->

    <record id="view\_creativeminds\_proyecto\_form" model="ir.ui.view">

        <field name="name">Nuevo proyecto</field>

        <field name="model">creativeminds.proyecto</field>

        <field name="arch" type="xml">

            <form string="Proyecto">

                <sheet>

                    <!-- Campos básicos del proyecto -->

                    <group>

                        <field name="proyecto\_id" />

                        <field name="nombre" />

                        <field name="estado"/>

                        <field name="empleado\_id" />

                        <field name="costo\_por\_hora" />

                        <field name="horas\_asignadas" />

                        <field name="costo\_total" readonly="1" />

                        <field name="descripcion" />

                        <field name="cliente" />

                    </group>

                    <!-- Más detalles del proyecto -->

                    <group>

                        <field name="porcentaje\_progreso" />

                        <field name="fecha\_inicio" />

                        <field name="fecha\_fin" />

                        <field name="prioridad" />

                        <field name="responsable\_id" />

                        <field name="presupuesto\_estimado" />

                        <field name="costo\_total\_recursos" readonly="1" />

                    </group>

                    <!-- Datos adicionales sobre tareas, recursos y riesgos -->

                    <group>

                        <field name="tareas\_ids"/>

                        <field name="recursos\_ids"/>

                        <field name="riesgos" />

                        <field name="hitos" />

                        <field name="dependencias" />

                        <field name="comentarios" />

                        <field name="recordatorios\_automaticos" />

                    </group>

                    <!-- Botones de acción para ver tareas, miembros, recursos, o duplicar proyecto -->

                    <group>

                        <p><button name="ver\_tareas" string="Ver Tareas" type="object" class="btn-secondary" context="{'proyecto\_id': proyecto\_id}" />

                        <button name="ver\_miembros" string="Ver Miembros" type="object" class="btn-secondary" context="{'proyecto\_id': proyecto\_id}" />

                        <button name="ver\_recursos" string="Ver Recursos" type="object" class="btn-secondary" context="{'proyecto\_id': proyecto\_id}" />

                        <button name="duplicar\_proyecto" string="Duplicar Proyecto" type="object" class="btn-primary" /></p>

                    </group>

                </sheet>

                <!-- Funcionalidad de conversación para seguimiento de mensajes -->

                <div class="oe\_chatter">

                    <field name="message\_follower\_ids" widget="mail\_followers" />

                    <field name="message\_ids" widget="mail\_thread" />

                </div>

            </form>

        </field>

    </record>

    <!-- Vista tipo árbol para listar proyectos -->

    <record id="view\_creativeminds\_proyecto\_tree" model="ir.ui.view">

        <field name="name">Lista de proyectos</field>

        <field name="model">creativeminds.proyecto</field>

        <field name="arch" type="xml">

            <tree string="Proyectos">

                <field name="proyecto\_id" />

                <field name="nombre" />

                <field name="estado" />

                <field name="fecha\_inicio" />

                <field name="fecha\_fin" />

                <field name="prioridad" />

                <field name="costo\_total" />

                <field name="costo\_total\_recursos" />

                <field name="responsable\_id" />

                <button name="ver\_tareas" string="Ver Tareas" type="object" />

                <button name="ver\_miembros" string="Ver Miembros" type="object" />

                <button name="ver\_recursos" string="Ver Recursos" type="object" />

            </tree>

        </field>

    </record>

    <!-- Vista tipo Kanban para proyectos -->

    <record id="view\_creativeminds\_proyecto\_kanban" model="ir.ui.view">

        <field name="name">Kanban proyectos</field>

        <field name="model">creativeminds.proyecto</field>

        <field name="arch" type="xml">

            <kanban js\_class="project\_kanban" quick\_create="1" create="1" edit="1" delete="1">

                <field name="id" />

                <field name="nombre" />

                <field name="estado" />

                <field name="fecha\_inicio" />

                <field name="fecha\_fin" />

                <field name="prioridad" />

                <field name="costo\_total" />

                <field name="costo\_total\_recursos" />

                <templates>

                    <t t-name="kanban-box">

                        <div class="o\_kanban\_card">

                            <strong><field name="nombre" /></strong>

                            <p><small>Estado: <field name="estado" /></small></p>

                            <p><small>Fecha de Inicio: <field name="fecha\_inicio" /></small></p>

                            <p><small>Fecha de Fin: <field name="fecha\_fin" /></small></p>

                            <p><small>Prioridad: <field name="prioridad" /></small></p>

                            <p><small>Costo Total: <field name="costo\_total" /></small></p>

                            <p><small>Costo Total de Recursos: <field name="costo\_total\_recursos" /></small></p>

                            <p><button name="ver\_tareas" string="Ver Tareas" type="object" class="btn-secondary" context="{'proyecto\_id': id}" />

                                <button name="ver\_miembros" string="Ver Miembros" type="object" class="btn-secondary" context="{'proyecto\_id': id}" /></p>

                            <div class="o\_kanban\_footer">

                                <p><button name="ver\_recursos" string="Ver Recursos" type="object" class="btn-secondary" context="{'proyecto\_id': proyecto\_id}" />

                                <button name="duplicar\_proyecto" string="Duplicar Proyecto" type="object" class="btn-primary" /></p>

                            </div>

                        </div>

                    </t>

                </templates>

            </kanban>

        </field>

    </record>

    <!-- Vista lista de control panel -->

    <record id="view\_creativeminds\_control\_panel\_tree" model="ir.ui.view">

        <field name="name">Panel de Control</field>

        <field name="model">creativeminds.control.panel</field>

        <field name="arch" type="xml">

            <tree>

                <field name="nombre" />

                <field name="proyectos\_ids" />

            </tree>

        </field>

    </record>

    <!-- Vista dashboard de control panel -->

    <record id="view\_creativeminds\_control\_panel\_dashboard" model="ir.ui.view">

        <field name="name">Panel de Control Dashboard</field>

        <field name="model">creativeminds.control.panel</field>

        <field name="arch" type="xml">

            <form create="false" edit="false" delete="false">

                <sheet>

                    <div class="o\_dashboard\_container">

                        <h1 class="text-center">Panel de Control de CreativeMinds</h1>

                        <!-- Botón para cargar datos desde la API -->

                        <div class="text-center" style="margin-bottom: 15px;">

                            <button name="action\_load\_data\_from\_api" type="object" string="Cargar Datos" class="btn-primary"/>

                        </div>

                        <div class="row mt16">

                            <div class="col-md-12">

                                <div class="alert alert-info text-center">

                                    <h3>Esperando conexión con API externa</h3>

                                    <p>Este panel mostrará automáticamente las métricas y KPIs cuando la API esté conectada.</p>

                                    <p>No se requiere la creación manual de registros ya que los datos se generan automáticamente basados en proyectos, empleados, tareas y equipos.</p>

                                </div>

                            </div>

                        </div>

                        <!-- Aquí irá el contenido del dashboard cuando la API esté conectada -->

                    </div>

                </sheet>

            </form>

        </field>

    </record>

    <!-- Vista lista de empleados -->

    <record id="view\_creativeminds\_employees\_tree" model="ir.ui.view">

        <field name="name">Lista de empleados</field>

        <field name="model">creativeminds.empleado</field>

        <field name="arch" type="xml">

            <tree string="Empleados">

                <field name="empleado\_id"/>

                <field name="name" />

                <field name="dni"/>

                <field name="phone"/>

                <field name="email" />

                <field name="fecha\_nacimiento" />

                <field name="fecha\_incorporacion" />

                <field name="puesto" />

                <field name="proyecto\_id" />

                <field name="tareas\_ids" />

                <field name="departamento" />

                <field name="puesto" />

            </tree>

        </field>

    </record>

    <!-- Vista formulario de empleados -->

    <record model="ir.ui.view" id="view\_creativeminds\_employee\_form">

        <field name="name">Nuevo empleado</field>

        <field name="model">creativeminds.empleado</field>

        <field name="arch" type="xml">

            <form >

                <sheet string ="Modelo para la gestión de empleadoss">

                    <div class = "oe\_title">

                        <h1>Empleado\_<field name="id"/></h1>

                    </div>

                    <div class = "row">

                        <div class = "oe\_title mt-4">

                        <h1>

                            <field name = "dni" placeholder="12345678Z" class="oe\_inline"/>

                        </h1>

                        </div>

                        <div class ="mt-4">

                        <field name="foto" widget="image" class="oe\_right" options="{'size':[0,200]}"/>

                        </div>

                    </div>

                    <!-- Tabla de datos generales del empleado -->

                    <separator string="Datos personales" colspan="4"/>

                    <group col="3">

                        <group >

                            <field name="name"/>

                            <field name="apellido1"/>

                            <field name="apellido2"/>

                        </group>

                        <group >

                            <field name="departamento" />

                            <field name="puesto" />

                            <field name="disponibilidad" />

                        </group>

                        <group >

                            <field name="fecha\_nacimiento"/>

                            <field name="fecha\_incorporacion"/>

                            <field name="partner\_id"/>

                        </group>

                    </group>

                    <separator string="Equipos" colspan="4"/>

                    <field name="equipo\_id" widget="many2many\_tags" can\_create="False" can\_write="False"/>

                    <separator string="Proyectos" colspan="4"/>

                    <field name="proyecto\_id" widget="many2many\_tags" can\_create="False" can\_write="False"/>

                    <separator string="Tareas" colspan="4"/>

                    <field name="tareas\_ids" widget="one2many"/>

                </sheet>

                <notebook>

                    <!-- Pestaña de historial de cambios o notas -->

                    <page string="Historial">

                        <field name="message\_ids" widget="mail\_thread" options="{'no\_create': True}" />

                    </page>

                </notebook>

            </form>

        </field>

    </record>

    <!-- Vistas y acciones para equipos, tareas y recursos -->

    <!-- Estas vistas permiten gestionar los equipos, tareas y recursos de los proyectos -->

    <!-- Vista de equipos -->

    <record id="view\_creativeminds\_teams\_tree" model="ir.ui.view">

        <field name="name">Lista de equipos</field>

        <field name="model">creativeminds.equipo</field>

        <field name="arch" type="xml">

            <tree string="Equipos">

                <field name="nombre" />

                <field name="empleado\_id"/>

                <field name="responsable\_id" />

                <field name="descripcion" />

                <field name="n\_miembros" />

            </tree>

        </field>

    </record>

    <!-- Vista de tareas -->

    <record id="view\_creativeminds\_task\_tree" model="ir.ui.view">

        <field name="name">Lista de tareas</field>

        <field name="model">creativeminds.tarea</field>

        <field name="arch" type="xml">

            <tree string="Tareas">

                <field name="proyecto\_id" />

                <field name="nombre" />

                <field name="descripcion" />

                <field name="responsable\_id" />

                <field name="fecha\_inicio" />

                <field name="fecha\_fin" />

                <field name="estado" />

            </tree>

        </field>

    </record>

    <!-- Vista de recursos -->

    <record id="view\_creativeminds\_resource\_tree" model="ir.ui.view">

        <field name="name">Lista de recursos</field>

        <field name="model">creativeminds.recurso</field>

        <field name="arch" type="xml">

            <tree string="Recursos">

                <field name="nombre" />

                <field name="empleado\_id" />

                <field name="proyecto\_id" />

                <field name="costo\_por\_hora" />

                <field name="horas\_asignadas" />

                <field name="costo\_total" />

                <field name="fecha\_inicio" />

                <field name="fecha\_fin" />

                <field name="estado" />

            </tree>

        </field>

    </record>

    <!-- Definición de acciones para mostrar las vistas en la interfaz de usuario -->

    <!-- Estas acciones permiten abrir las vistas de proyectos, empleados, tareas, recursos, ect -->

    <record id="action\_creativeminds\_proyecto" model="ir.actions.act\_window">

        <field name="name">Proyectos</field>

        <field name="res\_model">creativeminds.proyecto</field>

        <field name="view\_mode">tree,form,kanban</field>

    </record>

    <record id="action\_creativeminds\_employees" model="ir.actions.act\_window">

        <field name="name">Empleados</field>

        <field name="res\_model">creativeminds.empleado</field>

        <field name="view\_mode">tree,form</field>

    </record>

    <record id="action\_creativeminds\_teams" model="ir.actions.act\_window">

        <field name="name">Equipos</field>

        <field name="res\_model">creativeminds.equipo</field>

        <field name="view\_mode">tree,form</field>

    </record>

    <!-- Actualizar la acción para mostrar solo la vista de dashboard -->

    <record id="action\_creativeminds\_control\_panel" model="ir.actions.act\_window">

        <field name="name">Panel de Control</field>

        <field name="res\_model">creativeminds.control.panel</field>

        <field name="view\_mode">form</field>

        <field name="view\_id" ref="view\_creativeminds\_control\_panel\_dashboard"/>

        <field name="target">current</field>

        <field name="limit">1</field>

    </record>

    <record id="action\_creativeminds\_tasks" model="ir.actions.act\_window">

        <field name="name">Tareas</field>

        <field name="res\_model">creativeminds.tarea</field>

        <field name="view\_mode">tree,form</field>

    </record>

    <!-- Se deja por si si quiere usar en el futuro -->

    <record id="action\_creativeminds\_resources" model="ir.actions.act\_window">

        <field name="name">Recursos</field>

        <field name="res\_model">creativeminds.recurso</field>

        <field name="view\_mode">tree,form</field>

    </record>

    <!-- Menús para acceder a las distintas secciones del módulo -->

    <menuitem

        id="menu\_creativeminds\_proyecto\_root"

        name="CreativeMinds"

        web\_icon="creativeminds\_proyecto,static/description/icon.png"

        sequence="10"/>

    <!-- Submenús para proyectos, tareas -->

    <menuitem

        id="menu\_projects"

        name="Proyectos"

        parent="menu\_creativeminds\_proyecto\_root"

        action="action\_creativeminds\_proyecto"

        sequence="10"/>

    <menuitem

        id="menu\_projects\_projects"

        name="Ver proyectos"

        parent="menu\_projects"

        action="action\_creativeminds\_proyecto"

        sequence="10"/>

    <menuitem

        id="menu\_project\_tasks"

        name="Ver Tareas"

        parent="menu\_projects"

        action="action\_creativeminds\_tasks"

        sequence="20"/>

    <!-- Menús para empleados, equipos y panel de control -->

    <menuitem

        id="menu\_employees"

        name="Empleados"

        parent="menu\_creativeminds\_proyecto\_root"

        action="action\_creativeminds\_employees"

        sequence="20"/>

    <menuitem

        id="menu\_teams"

        name="Equipos"

        parent="menu\_creativeminds\_proyecto\_root"

        action="action\_creativeminds\_teams"

        sequence="30"/>

    <menuitem

        id="menu\_control\_panel"

        name="Panel de Control"

        parent="menu\_creativeminds\_proyecto\_root"

        action="action\_creativeminds\_control\_panel"

        sequence="40"/>

</odoo>

**creativeminds/security/ir.nodel.access.csv:**

# Acceso básico para usuario de proyecto: Permisos de lectura, escritura y creación, pero no de eliminación.

access\_creativeminds\_proyect\_user,access.creativeminds.proyecto.user,model\_creativeminds\_proyecto,base.group\_user,1,1,1,0

# Acceso completo para el gerente de proyecto: Permisos completos de lectura, escritura, creación y eliminación.

access\_creativeminds\_proyect\_manager,access.creativeminds.proyecto.manager,model\_creativeminds\_proyecto,project.group\_project\_manager,1,1,1,1

# Acceso básico para usuario de recursos: Permisos de lectura, escritura y creación, pero no de eliminación.

access\_creativeminds\_recurso\_user,access.creativeminds.recurso.user,model\_creativeminds.recurso,base.group\_user,1,1,1,0

# Acceso completo para el gerente de recursos: Permisos completos de lectura, escritura, creación y eliminación.

access\_creativeminds\_recurso\_manager,access.creativeminds.recurso.manager,model\_creativeminds.recurso,project.group\_project\_manager,1,1,1,1

# Acceso básico para usuario de tareas: Permisos de lectura, escritura y creación, pero no de eliminación.

access\_creativeminds\_tarea\_user,access.creativeminds.tarea.user,model\_creativeminds.tarea,base.group\_user,1,1,1,0

# Acceso completo para el gerente de tareas: Permisos completos de lectura, escritura, creación y eliminación.

access\_creativeminds\_tarea\_manager,access.creativeminds.tarea.manager,model\_creativeminds.tarea,project.group\_project\_manager,1,1,1,1

# Acceso básico para usuario de KPI: Permisos de lectura, escritura y creación, pero no de eliminación.

access\_creativeminds\_kpi\_user,access.creativeminds.kpi.user,model\_creativeminds.kpi,base.group\_user,1,1,1,0

# Acceso completo para el gerente de KPI: Permisos completos de lectura, escritura, creación y eliminación.

access\_creativeminds\_kpi\_manager,access.creativeminds.kpi.manager,model\_creativeminds.kpi,project.group\_project\_manager,1,1,1,1

# Acceso limitado para usuario de empleado: Solo lectura sin permisos de modificación ni eliminación.

access\_creativeminds\_empleado\_user,access.creativeminds.empleado.user,model\_creativeminds.empleado,base.group\_user,1,0,0,0

# Acceso completo para el gerente de empleados: Permisos completos de lectura, escritura, creación y eliminación.

access\_creativeminds\_empleado\_manager,access.creativeminds.empleado.manager,model\_creativeminds.empleado,hr.group\_hr\_manager,1,1,1,1

# Acceso limitado para usuario de equipo: Solo lectura sin permisos de modificación ni eliminación.

access\_creativeminds\_equipo\_user,access.creativeminds.equipo.user,model\_creativeminds.equipo,base.group\_user,1,0,0,0

# Acceso completo para el gerente de equipo: Permisos completos de lectura, escritura, creación y eliminación.

access\_creativeminds\_equipo\_manager,access.creativeminds.equipo.manager,model\_creativeminds.equipo,project.group\_project\_manager,1,1,1,1

# Acceso limitado para usuario de panel de control: Solo lectura sin permisos de modificación ni eliminación.

access\_creativeminds\_control\_panel\_user,access.creativeminds.control.panel.user,model\_creativeminds\_control\_panel,base.group\_user,1,0,0,0

# Acceso completo para el gerente de panel de control: Permisos completos de lectura, escritura, creación y eliminación.

access\_creativeminds\_control\_panel\_manager,access.creativeminds.control.panel.manager,model\_creativeminds\_control\_panel,project.group\_project\_manager,1,1,1,1

# Acceso limitado para usuario de métricas: Solo lectura sin permisos de modificación ni eliminación.

access\_creativeminds\_metrica\_user,access.creativeminds.metrica.user,model\_creativeminds.metrica,base.group\_user,1,0,0,0

# Acceso completo para el gerente de métricas: Permisos completos de lectura, escritura, creación y eliminación.

access\_creativeminds\_metrica\_manager,access.creativeminds.metrica.manager,model\_creativeminds.metrica,project.group\_project\_manager,1,1,1,1

# Acceso limitado para usuario de recomendaciones: Solo lectura sin permisos de modificación ni eliminación.

access\_creativeminds\_recomendacion\_user,access.creativeminds.recomendacion.user,model\_creativeminds.recomendacion,base.group\_user,1,0,0,0

# Acceso completo para el gerente de recomendaciones: Permisos completos de lectura, escritura, creación y eliminación.

access\_creativeminds\_recomendacion\_manager,access.creativeminds.recomendacion.manager,model\_creativeminds.recomendacion,project.group\_project\_manager,1,1,1,1

**creativeminds\_api/app.py:**

from flask import Flask, jsonify, request, Blueprint

from flask\_cors import CORS

import pandas as pd

import numpy as np

from datetime import datetime, timedelta

import json

from sqlalchemy import create\_engine

import logging

import os

from dotenv import load\_dotenv

# Configurar logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')

logger = logging.getLogger(\_\_name\_\_)

# Cargar variables de entorno

load\_dotenv()

app = Flask(\_\_name\_\_)

CORS(app)

# Configurar blueprints para organizar la API

api\_bp = Blueprint('api', \_\_name\_\_, url\_prefix='/api')

# Configuración de conexión a Odoo

def get\_odoo\_connection():

    try:

        db\_user = os.getenv('DB\_USER', 'admin')

        db\_password = os.getenv('DB\_PASSWORD', 'admin')

        db\_host = os.getenv('DB\_HOST', 'localhost')

        db\_port = os.getenv('DB\_PORT', '5432')

        db\_name = os.getenv('DB\_NAME', 'odoo')

        connection\_string = f"postgresql://{db\_user}:{db\_password}@{db\_host}:{db\_port}/{db\_name}"

        engine = create\_engine(connection\_string)

        return engine

    except Exception as e:

        logger.error(f"Error al conectar con la base de datos: {str(e)}")

        return None

# Rutas de la API

@api\_bp.route('/health', methods=['GET'])

def health\_check():

    """Endpoint para verificar que la API está funcionando"""

    return jsonify({"status": "OK", "message": "Creative Minds Analytics API está funcionando correctamente"})

@api\_bp.route('/dashboard', methods=['GET'])

def get\_dashboard():

    """Obtiene un resumen general del estado de todos los proyectos"""

    try:

        engine = get\_odoo\_connection()

        if not engine:

            return jsonify({"error": "No se pudo conectar a la base de datos"}), 500

        # Consulta para proyectos

        proyectos\_df = pd.read\_sql("""

            SELECT

                proyecto\_id, nombre, estado, fecha\_inicio, fecha\_fin,

                presupuesto\_estimado, costo\_total\_recursos, porcentaje\_progreso

            FROM

                creativeminds\_proyecto

        """, engine)

        # Consulta para tareas

        tareas\_df = pd.read\_sql("""

            SELECT

                t.id, t.nombre, t.estado, t.fecha\_comienzo, t.fecha\_final,

                t.proyecto\_id, p.nombre as nombre\_proyecto

            FROM

                creativeminds\_tarea t

            JOIN

                creativeminds\_proyecto p ON t.proyecto\_id = p.id

        """, engine)

        # Consulta para empleados

        empleados\_df = pd.read\_sql("""

            SELECT

                e.empleado\_id, e.nombre, e.disponibilidad, e.departamento, e.puesto

            FROM

                creativeminds\_empleado e

        """, engine)

        # Métricas generales

        metricas = {

            "total\_proyectos": len(proyectos\_df),

            "proyectos\_en\_progreso": len(proyectos\_df[proyectos\_df['estado'] == 'en\_progreso']),

            "proyectos\_finalizados": len(proyectos\_df[proyectos\_df['estado'] == 'finalizado']),

            "proyectos\_retrasados": \_calcular\_proyectos\_retrasados(proyectos\_df, tareas\_df),

            "progreso\_promedio": proyectos\_df['porcentaje\_progreso'].mean(),

            "presupuesto\_total": proyectos\_df['presupuesto\_estimado'].sum(),

            "costo\_actual\_total": proyectos\_df['costo\_total\_recursos'].sum(),

            "eficiencia\_presupuestaria": \_calcular\_eficiencia\_presupuestaria(proyectos\_df),

            "total\_tareas": len(tareas\_df),

            "tareas\_completadas": len(tareas\_df[tareas\_df['estado'] == 'completada']),

            "tareas\_pendientes": len(tareas\_df[tareas\_df['estado'] == 'pendiente']),

            "empleados\_disponibles": len(empleados\_df[empleados\_df['disponibilidad'] == 'disponible'])

        }

        # Proyectos principales (top 5 por progreso)

        top\_proyectos = proyectos\_df.sort\_values('porcentaje\_progreso', ascending=False).head(5)

        # Análisis de fortalezas y debilidades

        analisis = \_analizar\_fortalezas\_debilidades(proyectos\_df, tareas\_df, empleados\_df)

        # Recomendaciones para mejora

        recomendaciones = \_generar\_recomendaciones(proyectos\_df, tareas\_df, empleados\_df, analisis)

        return jsonify({

            "metricas": metricas,

            "proyectos\_destacados": top\_proyectos.to\_dict(orient='records'),

            "analisis": analisis,

            "recomendaciones": recomendaciones

        })

    except Exception as e:

        logger.error(f"Error en el dashboard: {str(e)}")

        return jsonify({"error": str(e)}), 500

@api\_bp.route('/proyectos', methods=['GET'])

def get\_proyectos():

    """Obtiene todos los proyectos con métricas detalladas"""

    try:

        engine = get\_odoo\_connection()

        if not engine:

            return jsonify({"error": "No se pudo conectar a la base de datos"}), 500

        proyectos\_df = pd.read\_sql("""

            SELECT

                p.\*,

                COUNT(DISTINCT t.id) as total\_tareas,

                SUM(CASE WHEN t.estado = 'completada' THEN 1 ELSE 0 END) as tareas\_completadas,

                COUNT(DISTINCT r.id) as total\_recursos

            FROM

                creativeminds\_proyecto p

            LEFT JOIN

                creativeminds\_tarea t ON p.id = t.proyecto\_id

            LEFT JOIN

                creativeminds\_recurso r ON p.id = r.proyecto\_id

            GROUP BY

                p.id

        """, engine)

        # Calcular métricas adicionales para cada proyecto

        proyectos\_metricas = []

        for \_, proyecto in proyectos\_df.iterrows():

            # Calcular eficiencia y estado general

            eficiencia = \_calcular\_eficiencia\_proyecto(proyecto)

            estado\_salud = \_determinar\_estado\_salud\_proyecto(proyecto)

            # Añadir métricas específicas

            proyecto\_data = proyecto.to\_dict()

            proyecto\_data["eficiencia"] = eficiencia

            proyecto\_data["estado\_salud"] = estado\_salud

            proyecto\_data["dias\_restantes"] = \_calcular\_dias\_restantes(proyecto)

            proyectos\_metricas.append(proyecto\_data)

        return jsonify({"proyectos": proyectos\_metricas})

    except Exception as e:

        logger.error(f"Error al obtener proyectos: {str(e)}")

        return jsonify({"error": str(e)}), 500

@api\_bp.route('/proyectos/<int:proyecto\_id>', methods=['GET'])

def get\_proyecto\_detalle(proyecto\_id):

    """Obtiene detalles completos de un proyecto específico con análisis profundo"""

    try:

        engine = get\_odoo\_connection()

        if not engine:

            return jsonify({"error": "No se pudo conectar a la base de datos"}), 500

        # Obtener datos del proyecto

        proyecto\_df = pd.read\_sql(f"""

            SELECT \* FROM creativeminds\_proyecto WHERE id = {proyecto\_id}

        """, engine)

        if proyecto\_df.empty:

            return jsonify({"error": "Proyecto no encontrado"}), 404

        # Obtener tareas relacionadas

        tareas\_df = pd.read\_sql(f"""

            SELECT \* FROM creativeminds\_tarea WHERE proyecto\_id = {proyecto\_id}

        """, engine)

        # Obtener recursos relacionados

        recursos\_df = pd.read\_sql(f"""

            SELECT \* FROM creativeminds\_recurso WHERE proyecto\_id = {proyecto\_id}

        """, engine)

        # Obtener KPIs relacionados

        kpis\_df = pd.read\_sql(f"""

            SELECT \* FROM creativeminds\_kpi WHERE proyecto\_id = {proyecto\_id}

        """, engine)

        # Calcular métricas específicas del proyecto

        metricas\_proyecto = \_calcular\_metricas\_proyecto(proyecto\_df.iloc[0], tareas\_df, recursos\_df, kpis\_df)

        # Generar análisis personalizado

        analisis\_proyecto = \_analizar\_proyecto(proyecto\_df.iloc[0], tareas\_df, recursos\_df, kpis\_df)

        # Generar recomendaciones específicas

        recomendaciones\_proyecto = \_generar\_recomendaciones\_proyecto(proyecto\_df.iloc[0], tareas\_df, recursos\_df)

        return jsonify({

            "proyecto": proyecto\_df.iloc[0].to\_dict(),

            "tareas": tareas\_df.to\_dict(orient='records'),

            "recursos": recursos\_df.to\_dict(orient='records'),

            "kpis": kpis\_df.to\_dict(orient='records'),

            "metricas": metricas\_proyecto,

            "analisis": analisis\_proyecto,

            "recomendaciones": recomendaciones\_proyecto

        })

    except Exception as e:

        logger.error(f"Error al obtener detalles del proyecto: {str(e)}")

        return jsonify({"error": str(e)}), 500

@api\_bp.route('/empleados', methods=['GET'])

def get\_empleados():

    """Obtiene datos de empleados con análisis de carga de trabajo y rendimiento"""

    try:

        engine = get\_odoo\_connection()

        if not engine:

            return jsonify({"error": "No se pudo conectar a la base de datos"}), 500

        # Obtener datos de empleados

        empleados\_df = pd.read\_sql("""

            SELECT

                e.\*,

                COUNT(DISTINCT pt.id) as total\_proyectos,

                COUNT(DISTINCT t.id) as total\_tareas,

                SUM(CASE WHEN t.estado = 'completada' THEN 1 ELSE 0 END) as tareas\_completadas

            FROM

                creativeminds\_empleado e

            LEFT JOIN

                creativeminds\_proyecto\_empleado\_rel pe ON e.id = pe.empleado\_id

            LEFT JOIN

                creativeminds\_proyecto pt ON pe.proyecto\_id = pt.id

            LEFT JOIN

                creativeminds\_tarea t ON t.responsable\_id = e.id

            GROUP BY

                e.id

        """, engine)

        # Calcular métricas por empleado

        empleados\_metricas = []

        for \_, empleado in empleados\_df.iterrows():

            empleado\_data = empleado.to\_dict()

            # Calcular tasa de completitud

            if empleado\_data["total\_tareas"] > 0:

                empleado\_data["tasa\_completitud"] = empleado\_data["tareas\_completadas"] / empleado\_data["total\_tareas"]

            else:

                empleado\_data["tasa\_completitud"] = 0

            # Calcular carga de trabajo

            empleado\_data["carga\_trabajo"] = \_calcular\_carga\_trabajo(empleado\_data)

            empleados\_metricas.append(empleado\_data)

        return jsonify({"empleados": empleados\_metricas})

    except Exception as e:

        logger.error(f"Error al obtener datos de empleados: {str(e)}")

        return jsonify({"error": str(e)}), 500

@api\_bp.route('/metricas/rendimiento', methods=['GET'])

def get\_metricas\_rendimiento():

    """Obtiene métricas de rendimiento general por departamento y equipo"""

    try:

        engine = get\_odoo\_connection()

        if not engine:

            return jsonify({"error": "No se pudo conectar a la base de datos"}), 500

        # Métricas por departamento

        departamentos\_df = pd.read\_sql("""

            SELECT

                e.departamento,

                COUNT(DISTINCT e.id) as total\_empleados,

                COUNT(DISTINCT p.id) as total\_proyectos,

                AVG(p.porcentaje\_progreso) as progreso\_promedio,

                SUM(p.costo\_total\_recursos) as costo\_total,

                SUM(p.presupuesto\_estimado) as presupuesto\_total

            FROM

                creativeminds\_empleado e

            LEFT JOIN

                creativeminds\_proyecto\_empleado\_rel pe ON e.id = pe.empleado\_id

            LEFT JOIN

                creativeminds\_proyecto p ON pe.proyecto\_id = p.id

            WHERE

                e.departamento IS NOT NULL

            GROUP BY

                e.departamento

        """, engine)

        # Métricas por equipo

        equipos\_df = pd.read\_sql("""

            SELECT

                eq.id, eq.nombre,

                COUNT(DISTINCT em.id) as total\_miembros,

                COUNT(DISTINCT p.id) as total\_proyectos,

                AVG(p.porcentaje\_progreso) as progreso\_promedio

            FROM

                creativeminds\_equipo eq

            LEFT JOIN

                creativeminds\_equipo\_empleado\_rel ee ON eq.id = ee.equipo\_id

            LEFT JOIN

                creativeminds\_empleado em ON ee.empleado\_id = em.id

            LEFT JOIN

                creativeminds\_proyecto\_empleado\_rel pe ON em.id = pe.empleado\_id

            LEFT JOIN

                creativeminds\_proyecto p ON pe.proyecto\_id = p.id

            GROUP BY

                eq.id, eq.nombre

        """, engine)

        # Añadir eficiencia por departamento

        for i, dept in departamentos\_df.iterrows():

            if dept['presupuesto\_total'] > 0:

                departamentos\_df.at[i, 'eficiencia\_presupuestaria'] = (

                    1 - (dept['costo\_total'] / dept['presupuesto\_total'])

                ) \* 100

            else:

                departamentos\_df.at[i, 'eficiencia\_presupuestaria'] = 0

        return jsonify({

            "departamentos": departamentos\_df.to\_dict(orient='records'),

            "equipos": equipos\_df.to\_dict(orient='records')

        })

    except Exception as e:

        logger.error(f"Error al obtener métricas de rendimiento: {str(e)}")

        return jsonify({"error": str(e)}), 500

@api\_bp.route('/metricas/historicas', methods=['GET'])

def get\_metricas\_historicas():

    """Obtiene métricas históricas para análisis de tendencias"""

    try:

        engine = get\_odoo\_connection()

        if not engine:

            return jsonify({"error": "No se pudo conectar a la base de datos"}), 500

        # Obtener proyectos con fechas

        proyectos\_df = pd.read\_sql("""

            SELECT

                id, nombre, estado, fecha\_inicio, fecha\_fin,

                presupuesto\_estimado, costo\_total\_recursos, porcentaje\_progreso

            FROM

                creativeminds\_proyecto

            WHERE

                fecha\_inicio IS NOT NULL

        """, engine)

        # Convertir fechas a formato datetime

        proyectos\_df['fecha\_inicio'] = pd.to\_datetime(proyectos\_df['fecha\_inicio'])

        proyectos\_df['fecha\_fin'] = pd.to\_datetime(proyectos\_df['fecha\_fin'])

        # Obtener el rango de tiempo (último año)

        hoy = datetime.now()

        inicio\_periodo = hoy - timedelta(days=365)

        # Filtrar proyectos del último año

        proyectos\_periodo = proyectos\_df[proyectos\_df['fecha\_inicio'] >= inicio\_periodo]

        # Crear series temporales por mes

        proyectos\_periodo['mes\_inicio'] = proyectos\_periodo['fecha\_inicio'].dt.to\_period('M')

        # Agrupar por mes y calcular métricas

        metricas\_mensuales = []

        for mes in pd.period\_range(start=inicio\_periodo, end=hoy, freq='M'):

            proyectos\_mes = proyectos\_periodo[proyectos\_periodo['mes\_inicio'] == mes]

            if not proyectos\_mes.empty:

                metricas\_mes = {

                    "mes": mes.strftime('%Y-%m'),

                    "proyectos\_iniciados": len(proyectos\_mes),

                    "presupuesto\_total": proyectos\_mes['presupuesto\_estimado'].sum(),

                    "costo\_total": proyectos\_mes['costo\_total\_recursos'].sum(),

                    "progreso\_promedio": proyectos\_mes['porcentaje\_progreso'].mean()

                }

                # Calcular eficiencia presupuestaria

                if metricas\_mes["presupuesto\_total"] > 0:

                    metricas\_mes["eficiencia\_presupuestaria"] = (

                        1 - (metricas\_mes["costo\_total"] / metricas\_mes["presupuesto\_total"])

                    ) \* 100

                else:

                    metricas\_mes["eficiencia\_presupuestaria"] = 0

                metricas\_mensuales.append(metricas\_mes)

        # Tendencias y predicciones

        tendencias = \_calcular\_tendencias(metricas\_mensuales)

        return jsonify({

            "metricas\_mensuales": metricas\_mensuales,

            "tendencias": tendencias

        })

    except Exception as e:

        logger.error(f"Error al obtener métricas históricas: {str(e)}")

        return jsonify({"error": str(e)}), 500

# Funciones auxiliares para cálculos y análisis

def \_calcular\_proyectos\_retrasados(proyectos\_df, tareas\_df):

    """Calcula el número de proyectos que están retrasados"""

    hoy = datetime.now().date()

    count = 0

    for \_, proyecto in proyectos\_df.iterrows():

        if proyecto['estado'] != 'finalizado' and proyecto['fecha\_fin'] and pd.to\_datetime(proyecto['fecha\_fin']).date() < hoy:

            count += 1

            continue

        # Verificar tareas retrasadas

        tareas\_proyecto = tareas\_df[tareas\_df['proyecto\_id'] == proyecto['id']]

        for \_, tarea in tareas\_proyecto.iterrows():

            if tarea['estado'] != 'completada' and tarea['fecha\_final'] and pd.to\_datetime(tarea['fecha\_final']).date() < hoy:

                count += 1

                break

    return count

def \_calcular\_eficiencia\_presupuestaria(proyectos\_df):

    """Calcula la eficiencia presupuestaria en porcentaje"""

    presupuesto\_total = proyectos\_df['presupuesto\_estimado'].sum()

    costo\_actual = proyectos\_df['costo\_total\_recursos'].sum()

    if presupuesto\_total > 0:

        return ((presupuesto\_total - costo\_actual) / presupuesto\_total) \* 100

    return 0

def \_calcular\_eficiencia\_proyecto(proyecto):

    """Calcula la eficiencia de un proyecto específico"""

    if proyecto['presupuesto\_estimado'] > 0:

        eficiencia\_presupuesto = (1 - (proyecto['costo\_total\_recursos'] / proyecto['presupuesto\_estimado'])) \* 100

    else:

        eficiencia\_presupuesto = 0

    if proyecto['total\_tareas'] > 0:

        eficiencia\_tareas = (proyecto['tareas\_completadas'] / proyecto['total\_tareas']) \* 100

    else:

        eficiencia\_tareas = 0

    # Ponderación: 60% presupuesto, 40% tareas

    return (eficiencia\_presupuesto \* 0.6) + (eficiencia\_tareas \* 0.4)

def \_determinar\_estado\_salud\_proyecto(proyecto):

    """Determina el estado de salud de un proyecto: Bueno, Regular, En riesgo, Crítico"""

    try:

        # Verificar si hay fecha de fin establecida y si está vencida

        hoy = datetime.now().date()

        fecha\_fin = pd.to\_datetime(proyecto['fecha\_fin']).date() if pd.notna(proyecto['fecha\_fin']) else None

        vencido = fecha\_fin and fecha\_fin < hoy and proyecto['estado'] != 'finalizado'

        # Calcular progreso esperado basado en fechas

        if fecha\_fin and pd.notna(proyecto['fecha\_inicio']):

            fecha\_inicio = pd.to\_datetime(proyecto['fecha\_inicio']).date()

            duracion\_total = (fecha\_fin - fecha\_inicio).days

            if duracion\_total > 0:

                tiempo\_transcurrido = (min(hoy, fecha\_fin) - fecha\_inicio).days

                progreso\_esperado = (tiempo\_transcurrido / duracion\_total) \* 100

                desviacion\_progreso = proyecto['porcentaje\_progreso'] - progreso\_esperado

            else:

                desviacion\_progreso = 0

        else:

            desviacion\_progreso = 0

        # Calcular desviación presupuestaria

        if proyecto['presupuesto\_estimado'] > 0:

            desviacion\_presupuesto = (proyecto['costo\_total\_recursos'] / proyecto['presupuesto\_estimado']) \* 100 - 100

        else:

            desviacion\_presupuesto = 0

        # Determinar estado de salud

        if vencido:

            return "Crítico"

        elif desviacion\_progreso < -20 or desviacion\_presupuesto > 20:

            return "En riesgo"

        elif desviacion\_progreso < -10 or desviacion\_presupuesto > 10:

            return "Regular"

        else:

            return "Bueno"

    except:

        return "No determinado"

def \_calcular\_dias\_restantes(proyecto):

    """Calcula los días restantes para la finalización del proyecto"""

    hoy = datetime.now().date()

    if pd.notna(proyecto['fecha\_fin']):

        fecha\_fin = pd.to\_datetime(proyecto['fecha\_fin']).date()

        dias\_restantes = (fecha\_fin - hoy).days

        return max(0, dias\_restantes)

    return None

def \_calcular\_metricas\_proyecto(proyecto, tareas\_df, recursos\_df, kpis\_df):

    """Calcula métricas detalladas para un proyecto específico"""

    hoy = datetime.now().date()

    # Cálculos básicos

    total\_tareas = len(tareas\_df)

    tareas\_completadas = len(tareas\_df[tareas\_df['estado'] == 'completada'])

    tareas\_pendientes = len(tareas\_df[tareas\_df['estado'] == 'pendiente'])

    tareas\_en\_progreso = len(tareas\_df[tareas\_df['estado'] == 'en\_progreso'])

    # Cálculos de tiempo

    fecha\_inicio = pd.to\_datetime(proyecto['fecha\_inicio']).date() if pd.notna(proyecto['fecha\_inicio']) else None

    fecha\_fin = pd.to\_datetime(proyecto['fecha\_fin']).date() if pd.notna(proyecto['fecha\_fin']) else None

    if fecha\_inicio and fecha\_fin:

        duracion\_total = (fecha\_fin - fecha\_inicio).days

        duracion\_transcurrida = (hoy - fecha\_inicio).days if hoy > fecha\_inicio else 0

        porcentaje\_tiempo\_transcurrido = (duracion\_transcurrida / duracion\_total) \* 100 if duracion\_total > 0 else 0

        desviacion\_tiempo = porcentaje\_tiempo\_transcurrido - proyecto['porcentaje\_progreso']

    else:

        duracion\_total = None

        duracion\_transcurrida = None

        porcentaje\_tiempo\_transcurrido = None

        desviacion\_tiempo = None

    # Cálculos de presupuesto

    if proyecto['presupuesto\_estimado'] > 0:

        porcentaje\_presupuesto\_usado = (proyecto['costo\_total\_recursos'] / proyecto['presupuesto\_estimado']) \* 100

        presupuesto\_restante = proyecto['presupuesto\_estimado'] - proyecto['costo\_total\_recursos']

    else:

        porcentaje\_presupuesto\_usado = 0

        presupuesto\_restante = 0

    # Valor ganado (EV) - método de gestión de proyectos

    valor\_planificado = proyecto['presupuesto\_estimado'] \* (porcentaje\_tiempo\_transcurrido / 100) if porcentaje\_tiempo\_transcurrido else 0

    valor\_ganado = proyecto['presupuesto\_estimado'] \* (proyecto['porcentaje\_progreso'] / 100)

    costo\_actual = proyecto['costo\_total\_recursos']

    indice\_rendimiento\_cronograma = valor\_ganado / valor\_planificado if valor\_planificado > 0 else 0

    indice\_rendimiento\_costo = valor\_ganado / costo\_actual if costo\_actual > 0 else 0

    return {

        "total\_tareas": total\_tareas,

        "tareas\_completadas": tareas\_completadas,

        "tareas\_pendientes": tareas\_pendientes,

        "tareas\_en\_progreso": tareas\_en\_progreso,

        "tasa\_completitud": (tareas\_completadas / total\_tareas \* 100) if total\_tareas > 0 else 0,

        "duracion\_total\_dias": duracion\_total,

        "dias\_transcurridos": duracion\_transcurrida,

        "porcentaje\_tiempo\_transcurrido": porcentaje\_tiempo\_transcurrido,

        "desviacion\_tiempo\_progreso": desviacion\_tiempo,

        "presupuesto\_estimado": proyecto['presupuesto\_estimado'],

        "costo\_actual": proyecto['costo\_total\_recursos'],

        "porcentaje\_presupuesto\_usado": porcentaje\_presupuesto\_usado,

        "presupuesto\_restante": presupuesto\_restante,

        "valor\_planificado": valor\_planificado,

        "valor\_ganado": valor\_ganado,

        "indice\_rendimiento\_cronograma": indice\_rendimiento\_cronograma,

        "indice\_rendimiento\_costo": indice\_rendimiento\_costo,

        "total\_recursos": len(recursos\_df),

        "total\_kpis": len(kpis\_df)

    }

def \_analizar\_proyecto(proyecto, tareas\_df, recursos\_df, kpis\_df):

    """Genera un análisis detallado de un proyecto específico"""

    metricas = \_calcular\_metricas\_proyecto(proyecto, tareas\_df, recursos\_df, kpis\_df)

    analisis = {

        "puntos\_fuertes": [],

        "puntos\_debiles": [],

        "riesgos": [],

        "oportunidades": []

    }

    # Analizar puntos fuertes

    if metricas["indice\_rendimiento\_costo"] > 1.05:

        analisis["puntos\_fuertes"].append("Excelente rendimiento de costos")

    if metricas["indice\_rendimiento\_cronograma"] > 1.05:

        analisis["puntos\_fuertes"].append("Progreso por encima del cronograma planificado")

    if metricas["tasa\_completitud"] > 75:

        analisis["puntos\_fuertes"].append("Alta tasa de completitud de tareas")

    if metricas["porcentaje\_presupuesto\_usado"] < 85 and metricas["porcentaje\_tiempo\_transcurrido"] > 80:

        analisis["puntos\_fuertes"].append("Eficiencia presupuestaria")

    # Analizar puntos débiles

    if metricas["indice\_rendimiento\_cronograma"] < 0.9:

        analisis["puntos\_debiles"].append("Retraso en el cronograma")

    if metricas["indice\_rendimiento\_costo"] < 0.9:

        analisis["puntos\_debiles"].append("Sobrecosto del proyecto")

    if metricas["porcentaje\_tiempo\_transcurrido"] > 70 and metricas["tasa\_completitud"] < 50:

        analisis["puntos\_debiles"].append("Baja tasa de completitud en relación al tiempo transcurrido")

    # Analizar riesgos

    if metricas["desviacion\_tiempo\_progreso"] > 20:

        analisis["riesgos"].append("Riesgo de retraso significativo en la entrega")

    if metricas["porcentaje\_presupuesto\_usado"] > 90 and metricas["porcentaje\_tiempo\_transcurrido"] < 80:

        analisis["riesgos"].append("Riesgo de sobrecosto del proyecto")

    tareas\_sin\_progreso = tareas\_df[(tareas\_df['estado'] == 'pendiente') & (pd.to\_datetime(tareas\_df['fecha\_comienzo']) < pd.Timestamp.now())]

    if len(tareas\_sin\_progreso) > 3:

        analisis["riesgos"].append(f"Hay {len(tareas\_sin\_progreso)} tareas que debieron iniciarse pero siguen pendientes")

    # Analizar oportunidades

    if len(kpis\_df) < 3:

        analisis["oportunidades"].append("Definir más KPIs para un mejor seguimiento del proyecto")

    if metricas["total\_recursos"] < 2:

        analisis["oportunidades"].append("Considerar asignar más recursos al proyecto")

    if metricas["tasa\_completitud"] > 95 and proyecto['estado'] != 'finalizado':

        analisis["oportunidades"].append("El proyecto está casi completado, considerar finalizarlo formalmente")

    # Si no hay nada, agregar mensajes predeterminados

    if not analisis["puntos\_fuertes"]:

        analisis["puntos\_fuertes"].append("No se identificaron puntos fuertes destacables")

    if not analisis["puntos\_debiles"]:

        analisis["puntos\_debiles"].append("No se identificaron puntos débiles significativos")

    if not analisis["riesgos"]:

        analisis["riesgos"].append("No se identificaron riesgos críticos en este momento")

    if not analisis["oportunidades"]:

        analisis["oportunidades"].append("Considerar realizar una revisión detallada para identificar oportunidades de mejora")

    return analisis

def \_generar\_recomendaciones\_proyecto(proyecto, tareas\_df, recursos\_df):

    """Genera recomendaciones específicas para mejorar un proyecto"""

    recomendaciones = []

    # Análisis de estado

    if proyecto['estado'] == 'en\_progreso':

        # Verificar si hay tareas sin asignar

        tareas\_sin\_responsable = tareas\_df[tareas\_df['responsable\_id'].isna()]

        if not tareas\_sin\_responsable.empty:

            recomendaciones.append(f"Asignar responsables a las {len(tareas\_sin\_responsable)} tareas sin asignar")

        # Verificar si hay tareas retrasadas

        hoy = datetime.now().date()

        tareas\_retrasadas = tareas\_df[(tareas\_df['estado'] != 'completada') &

                                      (pd.notnull(tareas\_df['fecha\_final'])) &

                                      (pd.to\_datetime(tareas\_df['fecha\_final']).dt.date < hoy)]

        if not tareas\_retrasadas.empty:

            recomendaciones.append(f"Priorizar las {len(tareas\_retrasadas)} tareas retrasadas")

    # Análisis de recursos

    if len(recursos\_df) == 0:

        recomendaciones.append("Asignar recursos al proyecto para un mejor seguimiento y control")

    # Verificar presupuesto

    if proyecto['costo\_total\_recursos'] > proyecto['presupuesto\_estimado'] \* 0.9:

        recomendaciones.append("Reevaluar el presupuesto del proyecto, ya que está cerca o por encima del límite")

    # Analizar progreso y fechas

    if pd.notnull(proyecto['fecha\_fin']):

        fecha\_fin = pd.to\_datetime(proyecto['fecha\_fin']).date()

        hoy = datetime.now().date()

        dias\_restantes = (fecha\_fin - hoy).days

        if dias\_restantes < 30 and proyecto['porcentaje\_progreso'] < 70:

            recomendaciones.append("Considerar extender la fecha de finalización o reasignar más recursos debido al bajo progreso")

    # Recomendaciones por estado

    if proyecto['estado'] == 'planificacion':

        recomendaciones.append("Definir hitos claros y KPIs medibles antes de iniciar el proyecto")

        recomendaciones.append("Realizar una evaluación de riesgos detallada")

    # Si el progreso es bajo

    if proyecto['porcentaje\_progreso'] < 25 and pd.notnull(proyecto['fecha\_inicio']):

        fecha\_inicio = pd.to\_datetime(proyecto['fecha\_inicio']).date()

        hoy = datetime.now().date()

        dias\_transcurridos = (hoy - fecha\_inicio).days

        if dias\_transcurridos > 30:

            recomendaciones.append("Evaluar los obstáculos que impiden el progreso del proyecto")

    # Si el proyecto tiene más de 10 tareas

    if len(tareas\_df) > 10:

        recomendaciones.append("Considerar dividir el proyecto en fases o subproyectos para un mejor seguimiento")

    # Si no hay recomendaciones, agregar una genérica

    if not recomendaciones:

        recomendaciones.append("El proyecto parece estar avanzando adecuadamente. Mantener el monitoreo regular")

    return recomendaciones

def \_analizar\_fortalezas\_debilidades(proyectos\_df, tareas\_df, empleados\_df):

    """Analiza fortalezas y debilidades globales de la gestión de proyectos"""

    analisis = {

        "fortalezas": [],

        "debilidades": [],

        "oportunidades": [],

        "amenazas": []

    }

    # Analizar proyectos

    proyectos\_en\_progreso = proyectos\_df[proyectos\_df['estado'] == 'en\_progreso']

    proyectos\_finalizados = proyectos\_df[proyectos\_df['estado'] == 'finalizado']

    # 1. Eficiencia presupuestaria

    if len(proyectos\_finalizados) > 0:

        eficiencia\_media = \_calcular\_eficiencia\_presupuestaria(proyectos\_finalizados)

        if eficiencia\_media > 10:

            analisis["fortalezas"].append(f"Buena eficiencia presupuestaria global ({eficiencia\_media:.2f}% de ahorro promedio)")

        elif eficiencia\_media < -5:

            analisis["debilidades"].append(f"Tendencia a superar presupuestos ({-eficiencia\_media:.2f}% de sobrecosto promedio)")

    # 2. Cumplimiento de plazos

    hoy = datetime.now().date()

    proyectos\_retrasados = sum(1 for \_, p in proyectos\_en\_progreso.iterrows()

                              if pd.notna(p['fecha\_fin']) and pd.to\_datetime(p['fecha\_fin']).date() < hoy)

    porcentaje\_retrasados = (proyectos\_retrasados / len(proyectos\_en\_progreso)) \* 100 if len(proyectos\_en\_progreso) > 0 else 0

    if porcentaje\_retrasados > 30:

        analisis["debilidades"].append(f"Alto porcentaje de proyectos retrasados ({porcentaje\_retrasados:.2f}%)")

    elif porcentaje\_retrasados < 10 and len(proyectos\_en\_progreso) > 3:

        analisis["fortalezas"].append(f"Excelente cumplimiento de plazos (solo {porcentaje\_retrasados:.2f}% de proyectos retrasados)")

    # 3. Disponibilidad de personal

    empleados\_disponibles = len(empleados\_df[empleados\_df['disponibilidad'] == 'disponible'])

    porcentaje\_disponibles = (empleados\_disponibles / len(empleados\_df)) \* 100 if len(empleados\_df) > 0 else 0

    if porcentaje\_disponibles < 15:

        analisis["amenazas"].append(f"Baja disponibilidad de personal ({porcentaje\_disponibles:.2f}%)")

    elif porcentaje\_disponibles > 40:

        analisis["oportunidades"].append(f"Alta disponibilidad de personal ({porcentaje\_disponibles:.2f}%) para nuevos proyectos")

    # 4. Progreso global

    progreso\_promedio = proyectos\_en\_progreso['porcentaje\_progreso'].mean() if len(proyectos\_en\_progreso) > 0 else 0

    if progreso\_promedio < 30:

        analisis["debilidades"].append(f"Bajo progreso promedio en los proyectos activos ({progreso\_promedio:.2f}%)")

    elif progreso\_promedio > 70:

        analisis["fortalezas"].append(f"Buen progreso promedio en los proyectos activos ({progreso\_promedio:.2f}%)")

    # 5. Diversidad de departamentos y equipos involucrados

    if 'departamento' in empleados\_df.columns:

        departamentos\_unicos = empleados\_df['departamento'].nunique()

        if departamentos\_unicos > 3:

            analisis["fortalezas"].append(f"Buena colaboración interdepartamental ({departamentos\_unicos} departamentos)")

    # 6. Evaluar la carga de trabajo

    tareas\_por\_empleado = []

    for \_, empleado in empleados\_df.iterrows():

        if 'id' in empleado:

            tareas = tareas\_df[tareas\_df['responsable\_id'] == empleado['id']]

            tareas\_por\_empleado.append(len(tareas))

    if tareas\_por\_empleado:

        max\_tareas = max(tareas\_por\_empleado) if tareas\_por\_empleado else 0

        desviacion\_tareas = np.std(tareas\_por\_empleado) if len(tareas\_por\_empleado) > 1 else 0

        if max\_tareas > 10:

            analisis["amenazas"].append(f"Posible sobrecarga de trabajo en algunos empleados (máx. {max\_tareas} tareas)")

        if desviacion\_tareas > 5:

            analisis["debilidades"].append("Distribución desigual de la carga de trabajo entre empleados")

    # Verificar que haya al menos un elemento en cada categoría

    if not analisis["fortalezas"]:

        analisis["fortalezas"].append("La organización mantiene operaciones estables")

    if not analisis["debilidades"]:

        analisis["debilidades"].append("No se identificaron debilidades críticas en este momento")

    if not analisis["oportunidades"]:

        analisis["oportunidades"].append("Considerar implementar más KPIs para medir el rendimiento de los proyectos")

    if not analisis["amenazas"]:

        analisis["amenazas"].append("Vigilar la asignación de recursos para evitar cuellos de botella")

    return analisis

def \_generar\_recomendaciones(proyectos\_df, tareas\_df, empleados\_df, analisis):

    """Genera recomendaciones para mejorar la gestión de proyectos basadas en el análisis"""

    recomendaciones = []

    # 1. Recomendaciones basadas en debilidades

    for debilidad in analisis.get("debilidades", []):

        if "presupuestos" in debilidad.lower():

            recomendaciones.append("Implementar un proceso más riguroso de estimación de presupuestos")

            recomendaciones.append("Establecer revisiones periódicas de gastos durante la ejecución de proyectos")

        elif "retrasados" in debilidad.lower():

            recomendaciones.append("Revisar y ajustar el proceso de planificación de plazos")

            recomendaciones.append("Implementar alertas tempranas para proyectos en riesgo de retraso")

        elif "progreso" in debilidad.lower():

            recomendaciones.append("Realizar revisiones semanales del progreso de proyectos críticos")

            recomendaciones.append("Considerar la implementación de metodologías ágiles para mejorar la velocidad de entrega")

        elif "distribución" in debilidad.lower() and "carga" in debilidad.lower():

            recomendaciones.append("Revisar la asignación de tareas para equilibrar la carga de trabajo")

            recomendaciones.append("Implementar un sistema de rotación para tareas repetitivas")

    # 2. Recomendaciones basadas en amenazas

    for amenaza in analisis.get("amenazas", []):

        if "disponibilidad" in amenaza.lower() and "personal" in amenaza.lower():

            recomendaciones.append("Evaluar la necesidad de contratar personal adicional o freelancers")

            recomendaciones.append("Priorizar proyectos y posiblemente posponer los menos críticos")

        elif "sobrecarga" in amenaza.lower():

            recomendaciones.append("Redistribuir tareas entre el equipo para evitar el agotamiento")

            recomendaciones.append("Considerar herramientas de automatización para tareas repetitivas")

    # 3. Recomendaciones basadas en oportunidades

    for oportunidad in analisis.get("oportunidades", []):

        if "disponibilidad" in oportunidad.lower() and "personal" in oportunidad.lower():

            recomendaciones.append("Aprovechar la disponibilidad para capacitar al personal en nuevas habilidades")

            recomendaciones.append("Considerar iniciar proyectos estratégicos planificados para el futuro")

    # 4. Recomendaciones específicas basadas en métricas

    # Verificar eficiencia presupuestaria global

    eficiencia\_global = \_calcular\_eficiencia\_presupuestaria(proyectos\_df)

    if eficiencia\_global < 0:

        recomendaciones.append("Realizar un análisis detallado de costos para identificar áreas de optimización")

    # Verificar proyectos sin KPIs definidos

    if 'indicadores\_ids' in proyectos\_df.columns:

        proyectos\_sin\_kpis = proyectos\_df[proyectos\_df['indicadores\_ids'].isna() | (proyectos\_df['indicadores\_ids'] == '[]')]

        if len(proyectos\_sin\_kpis) > 0:

            recomendaciones.append(f"Definir KPIs para los {len(proyectos\_sin\_kpis)} proyectos que carecen de indicadores")

    # Verificar proyectos casi finalizados

    proyectos\_casi\_terminados = proyectos\_df[(proyectos\_df['estado'] == 'en\_progreso') &

                                            (proyectos\_df['porcentaje\_progreso'] > 90)]

    if len(proyectos\_casi\_terminados) > 0:

        recomendaciones.append(f"Verificar criterios de finalización para {len(proyectos\_casi\_terminados)} proyectos con más del 90% de progreso")

    # Recomendaciones de mejores prácticas generales

    recomendaciones.append("Implementar reuniones retrospectivas al finalizar cada proyecto para identificar mejoras")

    recomendaciones.append("Mantener una base de conocimientos documentando lecciones aprendidas de cada proyecto")

    # Limitar a 10 recomendaciones máximo para no abrumar

    if len(recomendaciones) > 10:

        # Priorizar recomendaciones específicas sobre las generales

        recomendaciones = [r for r in recomendaciones if not r.startswith("Implementar") and not r.startswith("Mantener")][:8]

        recomendaciones.append("Implementar reuniones retrospectivas al finalizar cada proyecto")

        recomendaciones.append("Mantener una base de conocimientos con lecciones aprendidas")

    return recomendaciones

def \_calcular\_carga\_trabajo(empleado\_data):

    """Calcula el nivel de carga de trabajo de un empleado"""

    # Factores para considerar:

    # 1. Número de tareas asignadas

    # 2. Número de proyectos en los que participa

    # 3. Disponibilidad declarada

    nivel\_carga = 0

    # Factor de tareas

    if empleado\_data["total\_tareas"] > 15:

        nivel\_carga += 5

    elif empleado\_data["total\_tareas"] > 10:

        nivel\_carga += 4

    elif empleado\_data["total\_tareas"] > 7:

        nivel\_carga += 3

    elif empleado\_data["total\_tareas"] > 4:

        nivel\_carga += 2

    elif empleado\_data["total\_tareas"] > 1:

        nivel\_carga += 1

    # Factor de proyectos

    if empleado\_data["total\_proyectos"] > 5:

        nivel\_carga += 5

    elif empleado\_data["total\_proyectos"] > 3:

        nivel\_carga += 3

    elif empleado\_data["total\_proyectos"] > 1:

        nivel\_carga += 1

    # Factor de disponibilidad

    if empleado\_data["disponibilidad"] == "no\_disponible":

        nivel\_carga += 5

    elif empleado\_data["disponibilidad"] == "parcial":

        nivel\_carga += 3

    elif empleado\_data["disponibilidad"] == "asignado":

        nivel\_carga += 2

    # Normalizar a escala 0-10

    nivel\_carga = min(nivel\_carga, 10)

    # Determinar categoría de carga

    if nivel\_carga >= 8:

        return {"nivel": nivel\_carga, "categoria": "Alta"}

    elif nivel\_carga >= 5:

        return {"nivel": nivel\_carga, "categoria": "Media"}

    else:

        return {"nivel": nivel\_carga, "categoria": "Baja"}

def \_calcular\_tendencias(metricas\_mensuales):

    """Calcula tendencias a partir de las métricas históricas"""

    if not metricas\_mensuales or len(metricas\_mensuales) < 3:

        return {

            "tendencia\_proyectos": None,

            "tendencia\_presupuesto": None,

            "tendencia\_eficiencia": None,

            "prediccion\_proximos\_meses": None

        }

    # Extraer series temporales

    meses = [m["mes"] for m in metricas\_mensuales]

    proyectos\_por\_mes = [m["proyectos\_iniciados"] for m in metricas\_mensuales]

    presupuesto\_por\_mes = [m["presupuesto\_total"] for m in metricas\_mensuales]

    eficiencia\_por\_mes = [m.get("eficiencia\_presupuestaria", 0) for m in metricas\_mensuales]

    # Calcular tendencias (pendiente de la línea de regresión)

    x = np.arange(len(meses))

    # Tendencia de proyectos iniciados

    if len(set(proyectos\_por\_mes)) > 1:  # Verificar que hay variación

        slope\_proyectos = np.polyfit(x, proyectos\_por\_mes, 1)[0]

        tendencia\_proyectos = "creciente" if slope\_proyectos > 0.1 else "decreciente" if slope\_proyectos < -0.1 else "estable"

    else:

        tendencia\_proyectos = "estable"

    # Tendencia de presupuesto

    if len(set(presupuesto\_por\_mes)) > 1:  # Verificar que hay variación

        slope\_presupuesto = np.polyfit(x, presupuesto\_por\_mes, 1)[0]

        tendencia\_presupuesto = "creciente" if slope\_presupuesto > 100 else "decreciente" if slope\_presupuesto < -100 else "estable"

    else:

        tendencia\_presupuesto = "estable"

    # Tendencia de eficiencia

    if len(set(eficiencia\_por\_mes)) > 1:  # Verificar que hay variación

        slope\_eficiencia = np.polyfit(x, eficiencia\_por\_mes, 1)[0]

        tendencia\_eficiencia = "mejorando" if slope\_eficiencia > 0.5 else "empeorando" if slope\_eficiencia < -0.5 else "estable"

    else:

        tendencia\_eficiencia = "estable"

    # Predicción simple para próximos 3 meses (promedio móvil)

    ultimos\_proyectos = proyectos\_por\_mes[-3:] if len(proyectos\_por\_mes) >= 3 else proyectos\_por\_mes

    prediccion\_proyectos = round(sum(ultimos\_proyectos) / len(ultimos\_proyectos))

    ultimos\_presupuestos = presupuesto\_por\_mes[-3:] if len(presupuesto\_por\_mes) >= 3 else presupuesto\_por\_mes

    prediccion\_presupuesto = sum(ultimos\_presupuestos) / len(ultimos\_presupuestos)

    return {

        "tendencia\_proyectos": {

            "direccion": tendencia\_proyectos,

            "valores\_recientes": proyectos\_por\_mes[-3:] if len(proyectos\_por\_mes) >= 3 else proyectos\_por\_mes

        },

        "tendencia\_presupuesto": {

            "direccion": tendencia\_presupuesto,

            "valores\_recientes": [round(p, 2) for p in (presupuesto\_por\_mes[-3:] if len(presupuesto\_por\_mes) >= 3 else presupuesto\_por\_mes)]

        },

        "tendencia\_eficiencia": {

            "direccion": tendencia\_eficiencia,

            "valores\_recientes": [round(e, 2) for e in (eficiencia\_por\_mes[-3:] if len(eficiencia\_por\_mes) >= 3 else eficiencia\_por\_mes)]

        },

        "prediccion\_proximos\_meses": {

            "proyectos\_mensuales": prediccion\_proyectos,

            "presupuesto\_mensual": round(prediccion\_presupuesto, 2)

        }

    }

# Añadir endpoints personalizados

@api\_bp.route('/equipos', methods=['GET'])

def get\_equipos():

    """Obtiene información sobre los equipos de trabajo y su rendimiento"""

    try:

        engine = get\_odoo\_connection()

        if not engine:

            return jsonify({"error": "No se pudo conectar a la base de datos"}), 500

        # Obtener datos de equipos

        equipos\_df = pd.read\_sql("""

            SELECT

                e.id, e.nombre, e.descripcion, e.responsable\_id,

                COUNT(DISTINCT ee.empleado\_id) as num\_miembros

            FROM

                creativeminds\_equipo e

            LEFT JOIN

                creativeminds\_equipo\_empleado\_rel ee ON e.id = ee.equipo\_id

            GROUP BY

                e.id, e.nombre, e.descripcion, e.responsable\_id

        """, engine)

        # Obtener datos de miembros por equipo

        miembros\_df = pd.read\_sql("""

            SELECT

                ee.equipo\_id,

                e.empleado\_id,

                e.nombre,

                e.disponibilidad,

                e.departamento,

                e.puesto

            FROM

                creativeminds\_equipo\_empleado\_rel ee

            JOIN

                creativeminds\_empleado e ON ee.empleado\_id = e.id

        """, engine)

        # Obtener datos de proyectos por equipo

        proyectos\_equipo\_df = pd.read\_sql("""

            SELECT

                ee.equipo\_id,

                COUNT(DISTINCT pe.proyecto\_id) as total\_proyectos,

                AVG(p.porcentaje\_progreso) as progreso\_promedio

            FROM

                creativeminds\_equipo\_empleado\_rel ee

            JOIN

                creativeminds\_empleado e ON ee.empleado\_id = e.id

            JOIN

                creativeminds\_proyecto\_empleado\_rel pe ON e.id = pe.empleado\_id

            JOIN

                creativeminds\_proyecto p ON pe.proyecto\_id = p.id

            GROUP BY

                ee.equipo\_id

        """, engine)

        # Combinar datos

        equipos\_completos = []

        for \_, equipo in equipos\_df.iterrows():

            equipo\_data = equipo.to\_dict()

            # Obtener miembros del equipo

            miembros = miembros\_df[miembros\_df['equipo\_id'] == equipo['id']].to\_dict(orient='records')

            equipo\_data['miembros'] = miembros

            # Obtener datos de proyectos

            proyectos\_data = proyectos\_equipo\_df[proyectos\_equipo\_df['equipo\_id'] == equipo['id']]

            if not proyectos\_data.empty:

                equipo\_data['total\_proyectos'] = int(proyectos\_data['total\_proyectos'].iloc[0])

                equipo\_data['progreso\_promedio'] = float(proyectos\_data['progreso\_promedio'].iloc[0]) if pd.notna(proyectos\_data['progreso\_promedio'].iloc[0]) else 0

            else:

                equipo\_data['total\_proyectos'] = 0

                equipo\_data['progreso\_promedio'] = 0

            # Calcular rendimiento del equipo

            equipo\_data['rendimiento'] = \_calcular\_rendimiento\_equipo(equipo\_data, miembros)

            equipos\_completos.append(equipo\_data)

        return jsonify({"equipos": equipos\_completos})

    except Exception as e:

        logger.error(f"Error al obtener datos de equipos: {str(e)}")

        return jsonify({"error": str(e)}), 500

@api\_bp.route('/recursos', methods=['GET'])

def get\_recursos():

    """Obtiene información sobre los recursos asignados a los proyectos"""

    try:

        engine = get\_odoo\_connection()

        if not engine:

            return jsonify({"error": "No se pudo conectar a la base de datos"}), 500

        recursos\_df = pd.read\_sql("""

            SELECT

                r.\*,

                p.nombre as nombre\_proyecto

            FROM

                creativeminds\_recurso r

            LEFT JOIN

                creativeminds\_proyecto p ON r.proyecto\_id = p.id

        """, engine)

        # Calcular métricas adicionales

        recursos\_df['eficiencia\_costo'] = recursos\_df.apply(

            lambda r: (r['horas\_asignadas'] \* r['costo\_por\_hora']) / r['costo\_total']

            if r['costo\_total'] > 0 else 0,

            axis=1

        )

        return jsonify({"recursos": recursos\_df.to\_dict(orient='records')})

    except Exception as e:

        logger.error(f"Error al obtener datos de recursos: {str(e)}")

        return jsonify({"error": str(e)}), 500

@api\_bp.route('/predicciones', methods=['GET'])

def get\_predicciones():

    """Genera predicciones para la planificación futura"""

    try:

        engine = get\_odoo\_connection()

        if not engine:

            return jsonify({"error": "No se pudo conectar a la base de datos"}), 500

        # Obtener datos históricos

        proyectos\_df = pd.read\_sql("""

            SELECT

                id, nombre, estado, fecha\_inicio, fecha\_fin,

                presupuesto\_estimado, costo\_total\_recursos, porcentaje\_progreso,

                horas\_asignadas

            FROM

                creativeminds\_proyecto

            WHERE

                fecha\_inicio IS NOT NULL AND fecha\_fin IS NOT NULL

        """, engine)

        # Si no hay suficientes datos para predicciones

        if len(proyectos\_df) < 5:

            return jsonify({

                "error": "Datos insuficientes para generar predicciones confiables",

                "recomendacion": "Se necesitan al menos 5 proyectos completados para generar predicciones"

            })

        # Calcular promedios y tendencias

        proyectos\_finalizados = proyectos\_df[proyectos\_df['estado'] == 'finalizado']

        # Duración promedio de proyectos

        proyectos\_finalizados['fecha\_inicio'] = pd.to\_datetime(proyectos\_finalizados['fecha\_inicio'])

        proyectos\_finalizados['fecha\_fin'] = pd.to\_datetime(proyectos\_finalizados['fecha\_fin'])

        proyectos\_finalizados['duracion\_dias'] = (proyectos\_finalizados['fecha\_fin'] - proyectos\_finalizados['fecha\_inicio']).dt.days

        duracion\_promedio = proyectos\_finalizados['duracion\_dias'].mean()

        # Costo promedio por hora y por proyecto

        costo\_promedio\_hora = proyectos\_finalizados['costo\_por\_hora'].mean() if 'costo\_por\_hora' in proyectos\_finalizados.columns else None

        if costo\_promedio\_hora is None:

            costo\_promedio\_por\_proyecto = proyectos\_finalizados['costo\_total\_recursos'].mean()

        else:

            costo\_promedio\_por\_proyecto = None

        # Horas promedio por proyecto

        horas\_promedio = proyectos\_finalizados['horas\_asignadas'].mean() if 'horas\_asignadas' in proyectos\_finalizados.columns else None

        # Generar predicciones para nuevos proyectos

        predicciones = {

            "duracion\_estimada": {

                "promedio\_dias": round(duracion\_promedio, 1) if not pd.isna(duracion\_promedio) else None,

                "minima\_dias": round(proyectos\_finalizados['duracion\_dias'].min(), 1) if not proyectos\_finalizados.empty else None,

                "maxima\_dias": round(proyectos\_finalizados['duracion\_dias'].max(), 1) if not proyectos\_finalizados.empty else None

            },

            "costo\_estimado": {

                "promedio\_por\_proyecto": round(costo\_promedio\_por\_proyecto, 2) if costo\_promedio\_por\_proyecto is not None else None,

                "promedio\_por\_hora": round(costo\_promedio\_hora, 2) if costo\_promedio\_hora is not None else None,

                "horas\_promedio": round(horas\_promedio, 2) if horas\_promedio is not None else None

            },

            "capacidad\_optima": {

                "proyectos\_simultáneos": \_estimar\_capacidad\_optima(proyectos\_df),

                "recursos\_recomendados": \_estimar\_recursos\_recomendados(proyectos\_finalizados)

            }

        }

        return jsonify({"predicciones": predicciones})

    except Exception as e:

        logger.error(f"Error al generar predicciones: {str(e)}")

        return jsonify({"error": str(e)}), 500

# Funciones auxiliares adicionales

def \_calcular\_rendimiento\_equipo(equipo\_data, miembros):

    """Calcula una puntuación de rendimiento para un equipo basada en diversos factores"""

    if not miembros:

        return {"puntuacion": 0, "nivel": "No evaluable", "factores": []}

    factores = []

    puntuacion\_total = 0

    # Factor 1: Tamaño adecuado del equipo (óptimo entre 3-7 miembros)

    num\_miembros = len(miembros)

    if 3 <= num\_miembros <= 7:

        puntuacion\_total += 3

        factores.append({"nombre": "Tamaño óptimo", "puntuacion": 3})

    elif 2 <= num\_miembros < 3 or 7 < num\_miembros <= 10:

        puntuacion\_total += 2

        factores.append({"nombre": "Tamaño aceptable", "puntuacion": 2})

    else:

        puntuacion\_total += 1

        factores.append({"nombre": "Tamaño no óptimo", "puntuacion": 1})

    # Factor 2: Diversidad de habilidades (basado en departamentos/puestos)

    departamentos = set()

    puestos = set()

    for miembro in miembros:

        if 'departamento' in miembro and miembro['departamento']:

            departamentos.add(miembro['departamento'])

        if 'puesto' in miembro and miembro['puesto']:

            puestos.add(miembro['puesto'])

    diversidad = len(departamentos) + len(puestos)

    if diversidad >= 5:

        puntuacion\_total += 3

        factores.append({"nombre": "Alta diversidad", "puntuacion": 3})

    elif 3 <= diversidad < 5:

        puntuacion\_total += 2

        factores.append({"nombre": "Diversidad media", "puntuacion": 2})

    else:

        puntuacion\_total += 1

        factores.append({"nombre": "Baja diversidad", "puntuacion": 1})

    # Factor 3: Disponibilidad de los miembros

    disponibilidad\_puntos = 0

    for miembro in miembros:

        if 'disponibilidad' in miembro:

            if miembro['disponibilidad'] == 'disponible':

                disponibilidad\_puntos += 3

            elif miembro['disponibilidad'] == 'parcial':

                disponibilidad\_puntos += 2

            elif miembro['disponibilidad'] == 'asignado':

                disponibilidad\_puntos += 1

    disponibilidad\_promedio = disponibilidad\_puntos / num\_miembros if num\_miembros > 0 else 0

    if disponibilidad\_promedio > 2.5:

        puntuacion\_total += 3

        factores.append({"nombre": "Alta disponibilidad", "puntuacion": 3})

    elif disponibilidad\_promedio > 1.5:

        puntuacion\_total += 2

        factores.append({"nombre": "Disponibilidad media", "puntuacion": 2})

    else:

        puntuacion\_total += 1

        factores.append({"nombre": "Baja disponibilidad", "puntuacion": 1})

    # Factor 4: Productividad (basada en progreso de proyectos)

    if 'progreso\_promedio' in equipo\_data and equipo\_data['progreso\_promedio'] > 0:

        if equipo\_data['progreso\_promedio'] > 75:

            puntuacion\_total += 3

            factores.append({"nombre": "Alta productividad", "puntuacion": 3})

        elif equipo\_data['progreso\_promedio'] > 50:

            puntuacion\_total += 2

            factores.append({"nombre": "Productividad media", "puntuacion": 2})

        else:

            puntuacion\_total += 1

            factores.append({"nombre": "Productividad baja", "puntuacion": 1})

    else:

        # No hay datos de progreso

        puntuacion\_total += 1

        factores.append({"nombre": "Sin datos de productividad", "puntuacion": 1})

    # Calcular puntuación final (sobre 10)

    puntuacion\_max\_posible = 12  # 4 factores \* 3 puntos máx

    puntuacion\_final = (puntuacion\_total / puntuacion\_max\_posible) \* 10

    # Determinar nivel de rendimiento

    if puntuacion\_final >= 8:

        nivel = "Excelente"

    elif puntuacion\_final >= 6:

        nivel = "Bueno"

    elif puntuacion\_final >= 4:

        nivel = "Regular"

    else:

        nivel = "Bajo"

    return {

        "puntuacion": round(puntuacion\_final, 1),

        "nivel": nivel,

        "factores": factores

    }

def \_estimar\_capacidad\_optima(proyectos\_df):

    """Estima la capacidad óptima de proyectos simultáneos basada en datos históricos"""

    try:

        # Convertir fechas a datetime

        proyectos\_df['fecha\_inicio'] = pd.to\_datetime(proyectos\_df['fecha\_inicio'])

        proyectos\_df['fecha\_fin'] = pd.to\_datetime(proyectos\_df['fecha\_fin'])

        # Crear un rango de fechas para análisis

        if len(proyectos\_df) > 0:

            fecha\_min = proyectos\_df['fecha\_inicio'].min()

            fecha\_max = proyectos\_df['fecha\_fin'].max()

            if pd.notna(fecha\_min) and pd.notna(fecha\_max):

                # Crear serie temporal de proyectos activos por día

                dias = pd.date\_range(start=fecha\_min, end=fecha\_max)

                proyectos\_por\_dia = []

                for dia in dias:

                    # Contar proyectos activos en este día

                    activos = sum(1 for \_, p in proyectos\_df.iterrows()

                               if pd.notna(p['fecha\_inicio']) and pd.notna(p['fecha\_fin'])

                               and p['fecha\_inicio'] <= dia <= p['fecha\_fin'])

                    proyectos\_por\_dia.append(activos)

                if proyectos\_por\_dia:

                    # Calcular estadísticas

                    max\_proyectos = max(proyectos\_por\_dia)

                    promedio\_proyectos = sum(proyectos\_por\_dia) / len(proyectos\_por\_dia)

                    percentil\_75 = np.percentile(proyectos\_por\_dia, 75)

                    return {

                        "maximo\_historico": max\_proyectos,

                        "promedio\_historico": round(promedio\_proyectos, 1),

                        "recomendado": max(round(percentil\_75, 0), 1)  # Al menos 1 proyecto

                    }

        # Si no hay datos suficientes

        return {

            "maximo\_historico": None,

            "promedio\_historico": None,

            "recomendado": 2  # Valor predeterminado

        }

    except:

        return {

            "maximo\_historico": None,

            "promedio\_historico": None,

            "recomendado": 2  # Valor predeterminado

        }

def \_estimar\_recursos\_recomendados(proyectos\_df):

    """Estima la cantidad de recursos recomendados por proyecto basado en históricos"""

    if 'recursos\_ids' in proyectos\_df.columns:

        # Si tenemos datos directos de recursos

        try:

            recursos\_por\_proyecto = []

            for \_, proyecto in proyectos\_df.iterrows():

                if isinstance(proyecto['recursos\_ids'], list):

                    recursos\_por\_proyecto.append(len(proyecto['recursos\_ids']))

                elif isinstance(proyecto['recursos\_ids'], str) and proyecto['recursos\_ids'].startswith('['):

                    # Es un string que representa una lista

                    recursos\_por\_proyecto.append(len(json.loads(proyecto['recursos\_ids'])))

            if recursos\_por\_proyecto:

                promedio = sum(recursos\_por\_proyecto) / len(recursos\_por\_proyecto)

                return round(promedio, 1)

        except:

            pass

    # Si no hay datos o hay error, usar heurística basada en duración y presupuesto

    try:

        duracion\_promedio = proyectos\_df['duracion\_dias'].mean() if 'duracion\_dias' in proyectos\_df.columns else None

        presupuesto\_promedio = proyectos\_df['presupuesto\_estimado'].mean()

        if duracion\_promedio is not None and presupuesto\_promedio is not None:

            # Heurística simple: 1 recurso por cada 15 días de proyecto o por cada X presupuesto

            recursos\_por\_tiempo = duracion\_promedio / 15

            recursos\_por\_presupuesto = presupuesto\_promedio / 5000  # Ajustar según escala de presupuesto

            return round(max(recursos\_por\_tiempo, recursos\_por\_presupuesto), 1)

    except:

        pass

    # Valor predeterminado si no hay datos

    return 2

@api\_bp.route('/recomendaciones', methods=['GET'])

def get\_recomendaciones\_generales():

    """Genera recomendaciones generales para mejorar la gestión de proyectos"""

    try:

        engine = get\_odoo\_connection()

        if not engine:

            return jsonify({"error": "No se pudo conectar a la base de datos"}), 500

        # Obtener datos de proyectos

        proyectos\_df = pd.read\_sql("""

            SELECT

                proyecto\_id, nombre, estado, fecha\_inicio, fecha\_fin,

                presupuesto\_estimado, costo\_total\_recursos, porcentaje\_progreso

            FROM

                creativeminds\_proyecto

        """, engine)

        # Obtener tareas

        tareas\_df = pd.read\_sql("""

            SELECT

                t.id, t.nombre, t.estado, t.fecha\_comienzo, t.fecha\_final,

                t.proyecto\_id, p.nombre as nombre\_proyecto

            FROM

                creativeminds\_tarea t

            JOIN

                creativeminds\_proyecto p ON t.proyecto\_id = p.id

        """, engine)

        # Obtener empleados

        empleados\_df = pd.read\_sql("""

            SELECT

                e.empleado\_id, e.nombre, e.disponibilidad, e.departamento, e.puesto

            FROM

                creativeminds\_empleado e

        """, engine)

        # Análisis de fortalezas y debilidades

        analisis = \_analizar\_fortalezas\_debilidades(proyectos\_df, tareas\_df, empleados\_df)

        # Generar recomendaciones

        recomendaciones = \_generar\_recomendaciones(proyectos\_df, tareas\_df, empleados\_df, analisis)

        # Identificar áreas de mejora críticas

        areas\_mejora = \_identificar\_areas\_mejora(proyectos\_df, tareas\_df, empleados\_df)

        return jsonify({

            "analisis\_foda": analisis,

            "recomendaciones": recomendaciones,

            "areas\_mejora\_prioritarias": areas\_mejora

        })

    except Exception as e:

        logger.error(f"Error al generar recomendaciones: {str(e)}")

        return jsonify({"error": str(e)}), 500

def \_identificar\_areas\_mejora(proyectos\_df, tareas\_df, empleados\_df):

    """Identifica áreas críticas que necesitan mejora inmediata"""

    areas\_mejora = []

    # 1. Verificar proyectos críticos retrasados

    hoy = datetime.now().date()

    proyectos\_retrasados = []

    for \_, proyecto in proyectos\_df.iterrows():

        if proyecto['estado'] != 'finalizado' and pd.notna(proyecto['fecha\_fin']):

            fecha\_fin = pd.to\_datetime(proyecto['fecha\_fin']).date()

            if fecha\_fin < hoy:

                dias\_retraso = (hoy - fecha\_fin).days

                proyectos\_retrasados.append({

                    "id": proyecto['proyecto\_id'],

                    "nombre": proyecto['nombre'],

                    "dias\_retraso": dias\_retraso,

                    "progreso": proyecto['porcentaje\_progreso']

                })

    if proyectos\_retrasados:

        # Ordenar por días de retraso

        proyectos\_retrasados = sorted(proyectos\_retrasados, key=lambda x: x['dias\_retraso'], reverse=True)

        areas\_mejora.append({

            "area": "Proyectos retrasados",

            "descripcion": f"Hay {len(proyectos\_retrasados)} proyectos que han superado su fecha de finalización",

            "gravedad": "Alta" if len(proyectos\_retrasados) > 3 or (proyectos\_retrasados and proyectos\_retrasados[0]['dias\_retraso'] > 30) else "Media",

            "ejemplos": proyectos\_retrasados[:3],  # Top 3 más retrasados

            "accion\_recomendada": "Realizar reunión de revisión urgente con los responsables de estos proyectos"

        })

    # 2. Verificar sobrecarga de recursos

    if 'id' in empleados\_df.columns:

        empleados\_con\_tareas = {}

        for \_, tarea in tareas\_df.iterrows():

            if pd.notna(tarea['responsable\_id']):

                empleado\_id = tarea['responsable\_id']

                if empleado\_id not in empleados\_con\_tareas:

                    empleados\_con\_tareas[empleado\_id] = []

                empleados\_con\_tareas[empleado\_id].append(tarea)

        empleados\_sobrecargados = []

        for empleado\_id, tareas in empleados\_con\_tareas.items():

            if len(tareas) > 8:  # Umbral de sobrecarga

                # Buscar nombre del empleado

                nombre = "Desconocido"

                empleado\_info = empleados\_df[empleados\_df['id'] == empleado\_id]

                if not empleado\_info.empty:

                    nombre = empleado\_info['nombre'].iloc[0]

                empleados\_sobrecargados.append({

                    "id": empleado\_id,

                    "nombre": nombre,

                    "num\_tareas": len(tareas),

                    "tareas\_pendientes": sum(1 for t in tareas if t['estado'] == 'pendiente')

                })

        if empleados\_sobrecargados:

            areas\_mejora.append({

                "area": "Sobrecarga de recursos humanos",

                "descripcion": f"Hay {len(empleados\_sobrecargados)} empleados con excesiva carga de trabajo",

                "gravedad": "Alta" if len(empleados\_sobrecargados) > 3 else "Media",

                "ejemplos": sorted(empleados\_sobrecargados, key=lambda x: x['num\_tareas'], reverse=True)[:3],

                "accion\_recomendada": "Redistribuir tareas y considerar la asignación de más recursos"

            })

    # 3. Verificar proyectos con presupuesto excedido

    proyectos\_sobrecosto = []

    for \_, proyecto in proyectos\_df.iterrows():

        if proyecto['presupuesto\_estimado'] > 0 and proyecto['costo\_total\_recursos'] > proyecto['presupuesto\_estimado']:

            exceso = (proyecto['costo\_total\_recursos'] - proyecto['presupuesto\_estimado']) / proyecto['presupuesto\_estimado'] \* 100

            proyectos\_sobrecosto.append({

                "id": proyecto['proyecto\_id'],

                "nombre": proyecto['nombre'],

                "presupuesto": proyecto['presupuesto\_estimado'],

                "costo\_actual": proyecto['costo\_total\_recursos'],

                "exceso\_porcentaje": round(exceso, 1)

            })

    if proyectos\_sobrecosto:

        areas\_mejora.append({

            "area": "Control presupuestario",

            "descripcion": f"Hay {len(proyectos\_sobrecosto)} proyectos que han excedido su presupuesto",

            "gravedad": "Alta" if len(proyectos\_sobrecosto) > 2 else "Media",

            "ejemplos": sorted(proyectos\_sobrecosto, key=lambda x: x['exceso\_porcentaje'], reverse=True)[:3],

            "accion\_recomendada": "Realizar auditoría de costos y revisar procesos de estimación presupuestaria"

        })

    # 4. Verificar tareas bloqueadas o sin avance

    hoy = datetime.now().date()

    tareas\_estancadas = []

    for \_, tarea in tareas\_df.iterrows():

        if tarea['estado'] == 'en\_progreso' and pd.notna(tarea['fecha\_comienzo']):

            fecha\_inicio = pd.to\_datetime(tarea['fecha\_comienzo']).date()

            dias\_activa = (hoy - fecha\_inicio).days

            if dias\_activa > 14:  # Más de 2 semanas sin completarse

                tareas\_estancadas.append({

                    "id": tarea['id'],

                    "nombre": tarea['nombre'],

                    "proyecto": tarea['nombre\_proyecto'],

                    "dias\_activa": dias\_activa

                })

    if tareas\_estancadas:

        areas\_mejora.append({

            "area": "Progreso de tareas",

            "descripcion": f"Hay {len(tareas\_estancadas)} tareas en progreso por más de 2 semanas",

            "gravedad": "Media",

            "ejemplos": sorted(tareas\_estancadas, key=lambda x: x['dias\_activa'], reverse=True)[:3],

            "accion\_recomendada": "Revisar bloqueos y dependencias que impiden el avance de las tareas"

        })

    return areas\_mejora

# Registrar los blueprints

app.register\_blueprint(api\_bp)

# Ejecutar la aplicación

if \_\_name\_\_ == '\_\_main\_\_':

    # Configurar logging

    logger = logging.getLogger(\_\_name\_\_)

    logger.setLevel(logging.INFO)

    # Configurar manejador para la consola

    console\_handler = logging.StreamHandler()

    console\_handler.setLevel(logging.INFO)

    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

    console\_handler.setFormatter(formatter)

    logger.addHandler(console\_handler)

    # Configurar manejador para archivo de log

    try:

        file\_handler = logging.FileHandler('creativeminds\_api.log')

        file\_handler.setLevel(logging.INFO)

        file\_handler.setFormatter(formatter)

        logger.addHandler(file\_handler)

    except:

        logger.warning("No se pudo crear el archivo de log")

    logger.info("Iniciando Creative Minds Analytics API...")

    # Ejecutar en modo debug y permitir acceso desde cualquier host

    app.run(debug=True, host='0.0.0.0', port=5000)