Documentación Técnica: Módulos Odoo y Web Controllers

Autor: Pedro José Meixús Belsol

2 de febrero de 2026

Índice

1. Modificar módulo liga fútbol

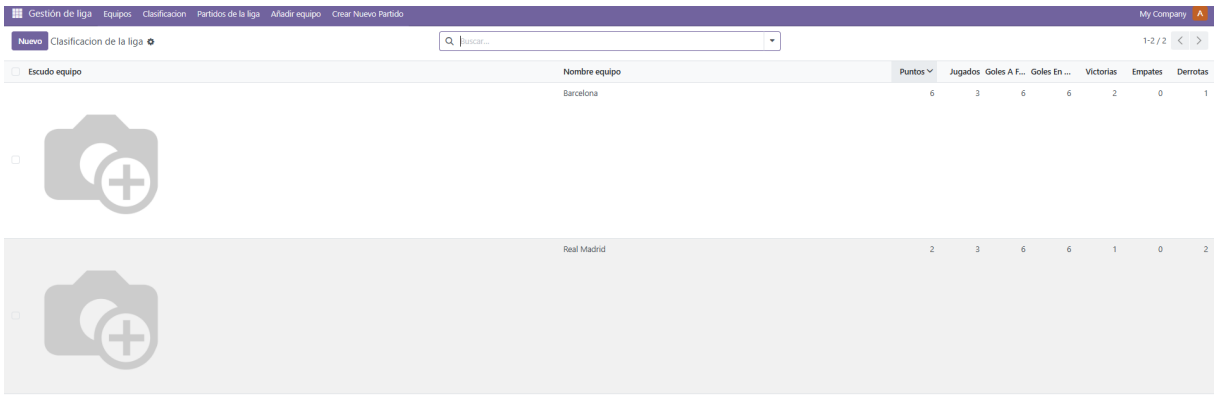
Tomando como base el módulo `EJ07LigaFutbol` que se nos ha proporcionado, vamos a realizar diferentes modificaciones para ampliar su funcionalidad.

1.1. Reglas de puntuación especiales

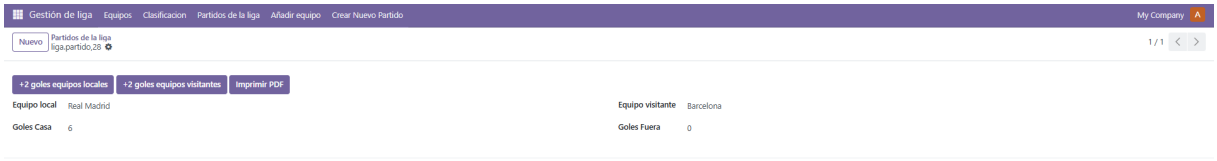
Empezaremos por modificar las reglas de puntuación. Haremos que los partidos con 4 o más goles de diferencia otorguen 4 puntos al equipo ganador y -1 al perdedor. Para eso crearemos una función en el modelo de 'liga_partido.py' para que lo calcule, simplemente comparando la diferencia de goles de goles.

```python
def actualizoRegistrosEquipo(self):
    for recordEquipo in self.env['liga.equipo'].search([]):
        recordEquipo.victorias = 0
        recordEquipo.empates = 0
        recordEquipo.derrotas = 0
        recordEquipo.goles_a_favor = 0
        recordEquipo.goles_en_contra = 0
        recordEquipo.puntos = 0

    for partido in self.env['liga.partido'].search([]):
        # Local
        local = partido.equipo_casa
        visitante = partido.equipo_fuera
        if not local or not visitante:
            continue

        # Goles
        goles_local = partido.goles_casa
        goles_vis = partido.goles_fuera
        diferencia = abs(goles_local - goles_vis)

        # Resultado local
        if goles_local > goles_vis:
            if diferencia >= 4:
                local.puntos += 4
            else:
                local.puntos += 3
            local.victorias += 1
            visitante.derrotas += 1
            if diferencia >= 4:
                visitante.puntos -= 1
        elif goles_local < goles_vis:
            if diferencia >= 4:
                visitante.puntos += 4
                local.puntos -= 1
            else:
                visitante.puntos += 3
            visitante.victorias += 1
            local.derrotas += 1
        else:
            local.empates += 1
            visitante.empates += 1
            local.puntos += 1
            visitante.puntos += 1

        # Sumar goles a favor y en contra
        local.goles_a_favor += goles_local
        local.goles_en_contra += goles_vis
        visitante.goles_a_favor += goles_vis
        visitante.goles_en_contra += goles_local
```

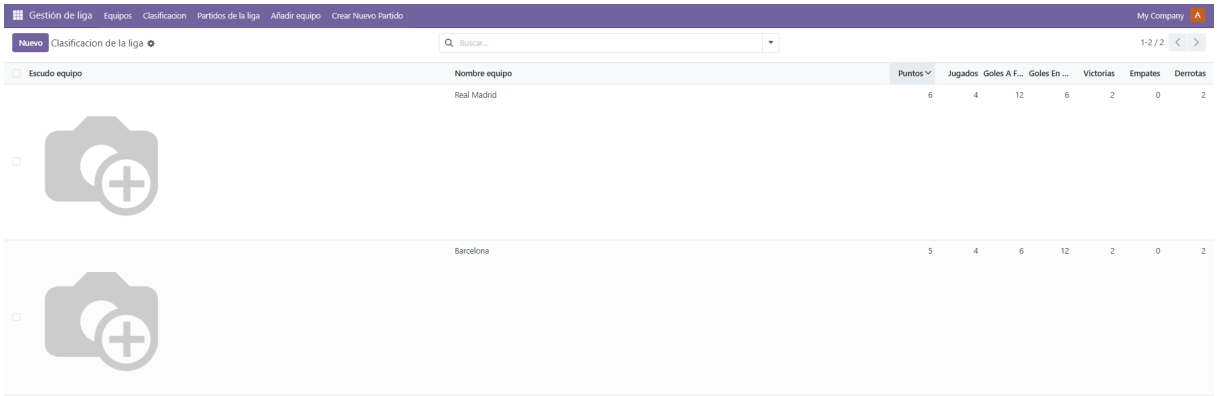
Podemos comprobarlo creando un partido con 4 goles de diferencia y viendo la tabla de clasificación. Ahora mismo esta es la tabla de clasificación:



Vamos a añadir un partido en el que el equipo Real Madrid gana 6-0.



Y ahora la puntuación debería haber aumentando en 4 puntos para el Real Madrid y disminuido en 1 para el Barcelona.

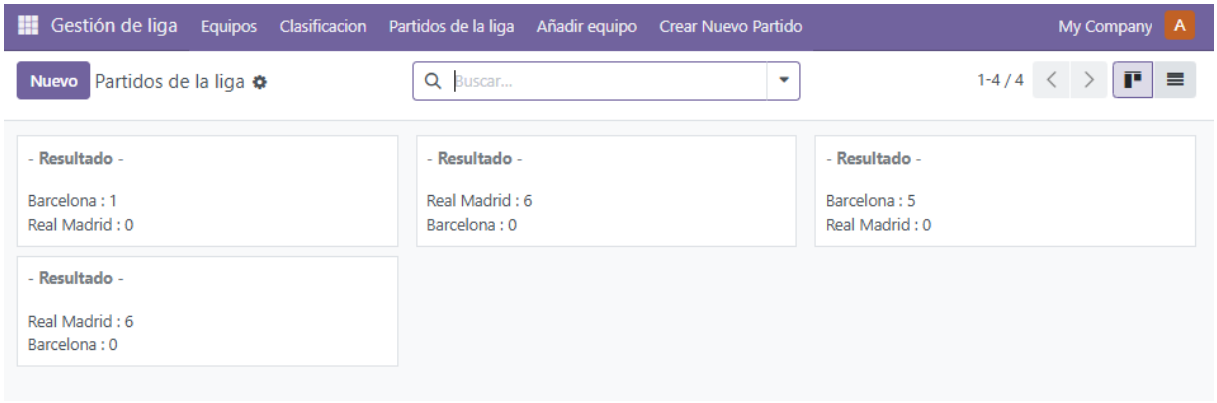


1.2. Botones para alterar los goles de los partidos

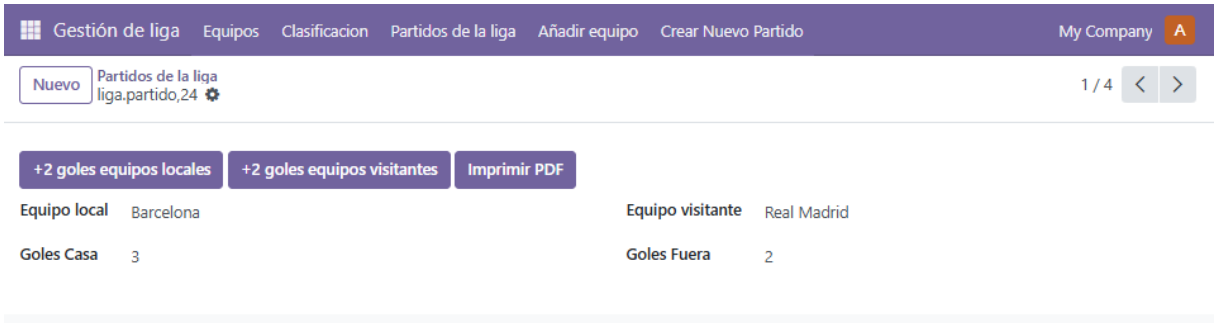
Vamos a añadir dos botones en la vista de formulario de los partidos para aumentar o disminuir los goles de todos los equipos locales o visitantes. Tendremos que añadir dos funciones en el modelo de 'liga_partido.py' para que lo hagan.

```python
    def sumar_goles_locales(self):
        for partido in self.env['liga.partido'].search([]):
            partido.goles_casa += 2
        self.actualizoRegistrosEquipo()

    def sumar_goles_visitantes(self):
        for partido in self.env['liga.partido'].search([]):
            partido.goles_fuera += 2
        self.actualizoRegistrosEquipo()
```

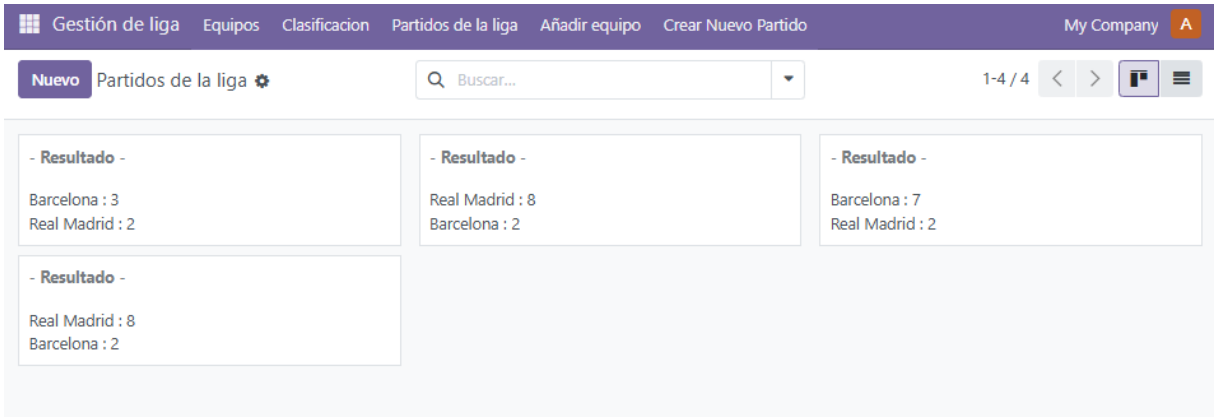
Vamos a probar si funciona, primero veremos los resultados de los partidos.



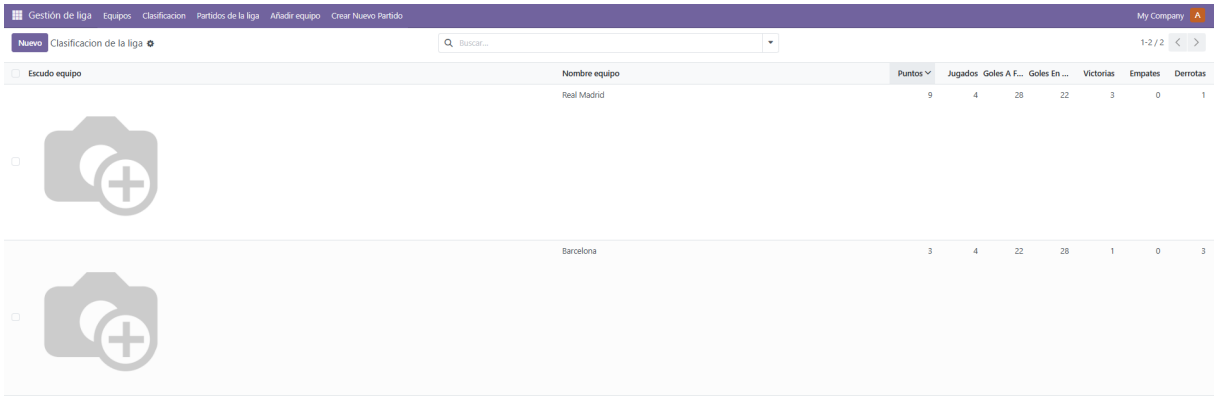
Ahora accedo a uno de ellos y pulso una vez cada botón.



Y ahora volvemos a ver los resultados de los partidos para comprobar que se han modificado correctamente.



Para probar que la clasificación se recalcula, voy a sumarle dos veces a los visitantes y al ver la clasificación podemos ver que ha cambiado.

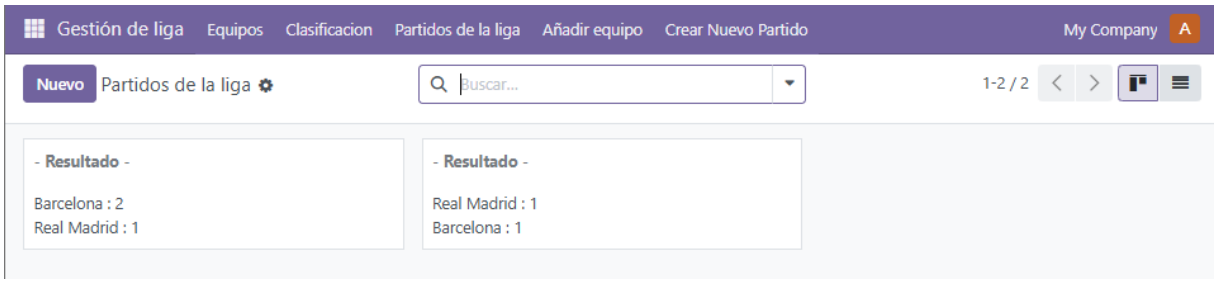


1.3. Web Controller para eliminar empates

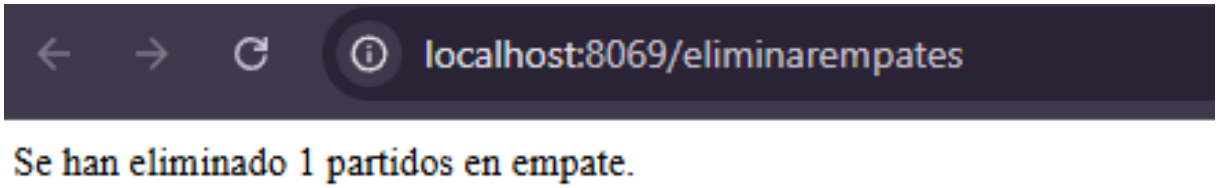
Vamos a añadir un Web Controller que nos permita eliminar todos los partidos que hayan terminado en empate. Primero añadiremos un '@http.route' con una función que buscara los partidos terminados en empate y los eliminará.

```python
@http.route('/eliminarempates', type='http', auth='none')
def eliminar_empates(self):
    # Buscamos todos los partidos que hayan terminado en empate
    partidos_empate = request.env['liga.partido'].sudo().search([
        ('goles_casa', '=', 'goles_fuera')
    ])

    # Contamos cuántos hay antes de eliminar
    total_eliminados = len(partidos_empate)

    # Eliminamos los partidos encontrados
    partidos_empate.unlink()

    # Devolvemos el número de partidos eliminados como texto
    return f"Se han eliminado {total_eliminados} partidos en empate."
```

Ahora crearemos un partido que termine en empate, si no tenemos alguno lo crearemos como en mi caso.



Teniendo uno por lo menos iremos a la URL del Web Controller que hemos creado ('http://localhost:8069/eliminarempates') y eliminará automaticamente los partidos en empate.



Y si volvemos a la vista de partidos podemos ver que el partido en empate ha sido eliminado.


Gestión de liga
Equipos
Clasificacion
Partidos de la liga
Añadir equipo
Crear Nuevo Partido
Nuevo
Partidos de la liga
Buscar...
- Resultado -
Barcelona : 2
Real Madrid : 1

1.4. Informe PDF por partido

No he conseguido hacer funcionar esta parte.

*Revisar en la sección de problemas encontrados al final del documento.

1.5. Wizard para crear nuevos partidos

Vamos a crear un Wizard que nos permita crear nuevos partidos de una forma más sencilla. Para eso crearemos dos archivos nuevos, uno para el modelo del Wizard y otro para la vista dentro de la carpeta 'wizard'.

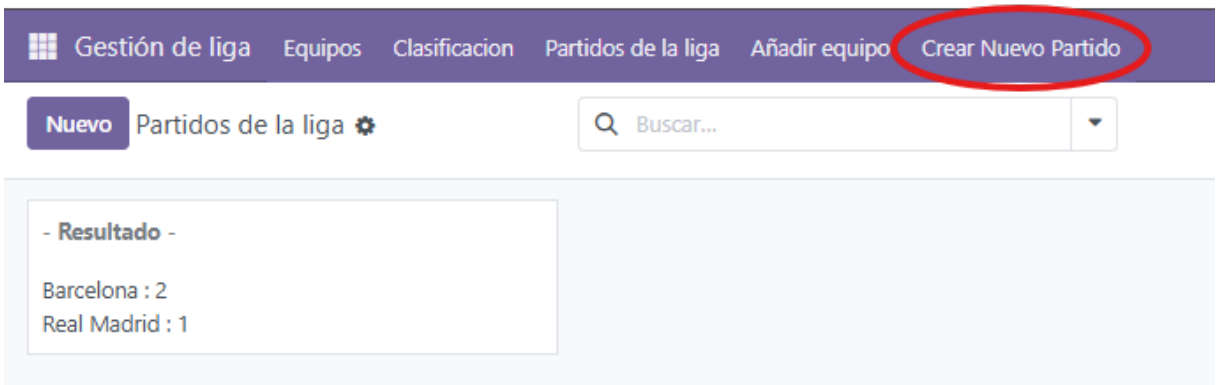
En el modelo crearemos los campos necesarios (y sus relaciones) para crear un partido y una función que lo cree.

```python
from odoo import models, fields, api

class LigaPartidoWizard(models.TransientModel):
    _name = 'wizard.liga.partido'
    _description = 'Wizard para crear un nuevo partido'

    equipo_casa = fields.Many2one('liga.equipo', string='Equipo Local', required=True)
    goles_casa = fields.Integer(string='Goles Local', default=0)
    equipo_fuera = fields.Many2one('liga.equipo', string='Equipo Visitante', required=True)
    goles_fuera = fields.Integer(string='Goles Visitante', default=0)
    jornada = fields.Integer(string='Jornada', required=True)

    def crear_partido(self):
        """Crea un nuevo partido y actualiza la clasificación automáticamente"""
        self.env['liga.partido'].create({
            'equipo_casa': self.equipo_casa.id,
            'goles_casa': self.goles_casa,
            'equipo_fuera': self.equipo_fuera.id,
            'goles_fuera': self.goles_fuera,
        })
        # Se cierra el wizard automáticamente
        return {'type': 'ir.actions.act_window_close'}
```

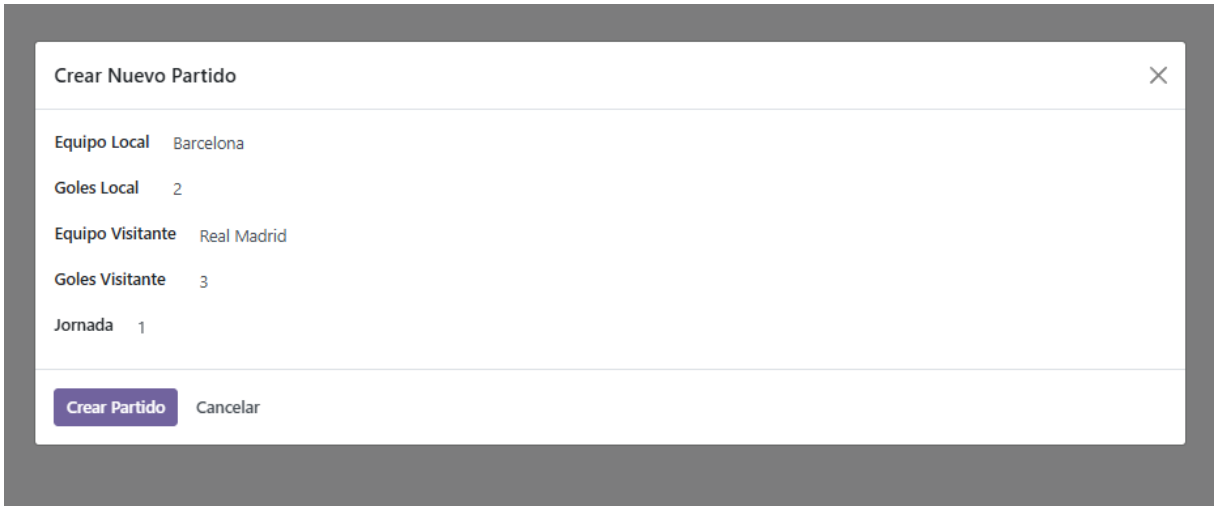
Y en la vista crearemos el formulario para el Wizard.

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>

    <!-- Vista del wizard -->
    <record id='liga_partido_wizard_form' model='ir.ui.view'>
        <field name='name'>Wizard para introducir un partido</field>
        <field name='model'>wizard.liga.partido</field>
        <field name='arch' type='xml'>
            <form string="Introducir un nuevo partido">
                <sheet>
                    <group>
                        <field name='equipo_casa'/>
                        <field name='goles_casa'/>
                    </group>
                    <group>
                        <field name='equipo_fuera'/>
                        <field name='goles_fuera'/>
                    </group>
                    <group>
                        <field name='jornada'/>
                    </group>
                </sheet>
                <footer>
                    <button string='Crear Partido' name='crear_partido' class='btn-primary' type='object'/>
                    <button string='Cancelar' class='btn-default' special='cancel'/>
                </footer>
            </form>
        </field>
    </record>

    <!-- Acción del wizard -->
    <record id="action_wizard_liga_partido" model="ir.actions.act_window">
        <field name="name">Crear Nuevo Partido</field>
        <field name="res_model">wizard.liga.partido</field>
        <field name="view_mode">form</field>
        <field name="target">new</field>
    </record>

    <!-- Menuitem -->
    <menuitem id="menu_wizard_liga_partido"
              parent="liga_base_menu"
              action="action_wizard_liga_partido"
              sequence="20"
              name="Crear Nuevo Partido"/>
</odoo>
```

Con esto ya hecho, añadiremos la ruta del xml al '__manifest__.py' para que Odoo los reconozca. Y ahora veremos un nuevo botón en el menú superior para abrir el Wizard.



Al pulsarlo se abrirá el Wizard que hemos creado y podremos añadir un nuevo partido.



1.6. Vista Graph para goles de equipos locales

Vamos a añadir una vista Graph que nos muestre los goles totales de los equipos locales.

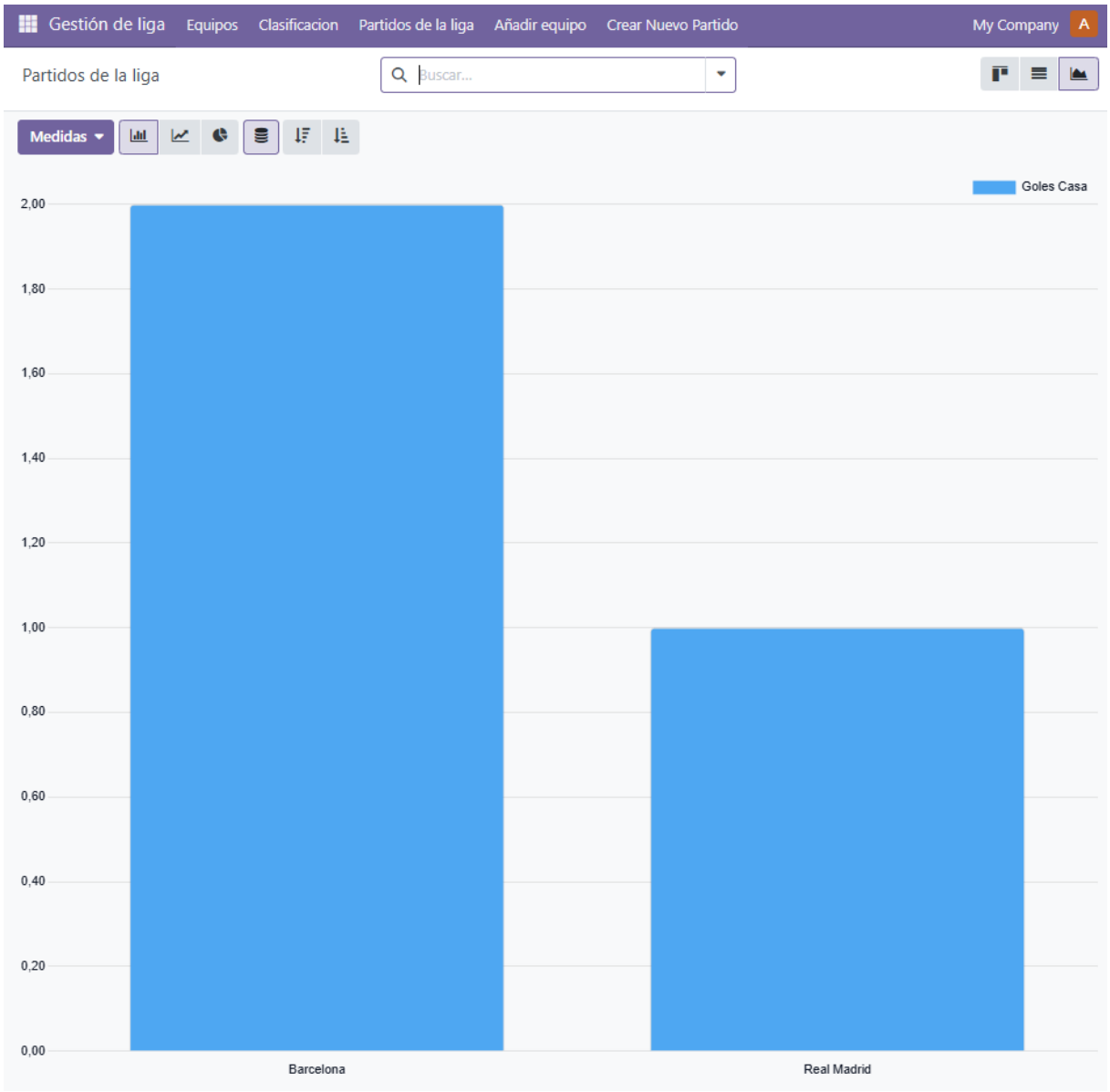
Para esto he añadido la vista Graph en el archivo XML de 'liga_partido.xml'.

```xml
<record id="liga_partido_view_graph" model="ir.ui.view">
    <field name="name">Goles locales por equipo</field>
    <field name="model">liga.partido</field>
    <field name="type">graph</field>
    <field name="arch" type="xml">
        <graph string="Goles locales por equipo">
            <field name="equipo_casa" type="row"/>
            <field name="goles_casa" type="measure"/>
        </graph>
    </field>
</record>
```

Y también he añadido la acción para que se una de las vistas posibles.

```xml
<field name="view_mode">kanban,list,form,graph</field>
```

Ahora si vamos a la vista de partidos podemos seleccionar la vista Graph y ver los goles totales de los equipos locales.

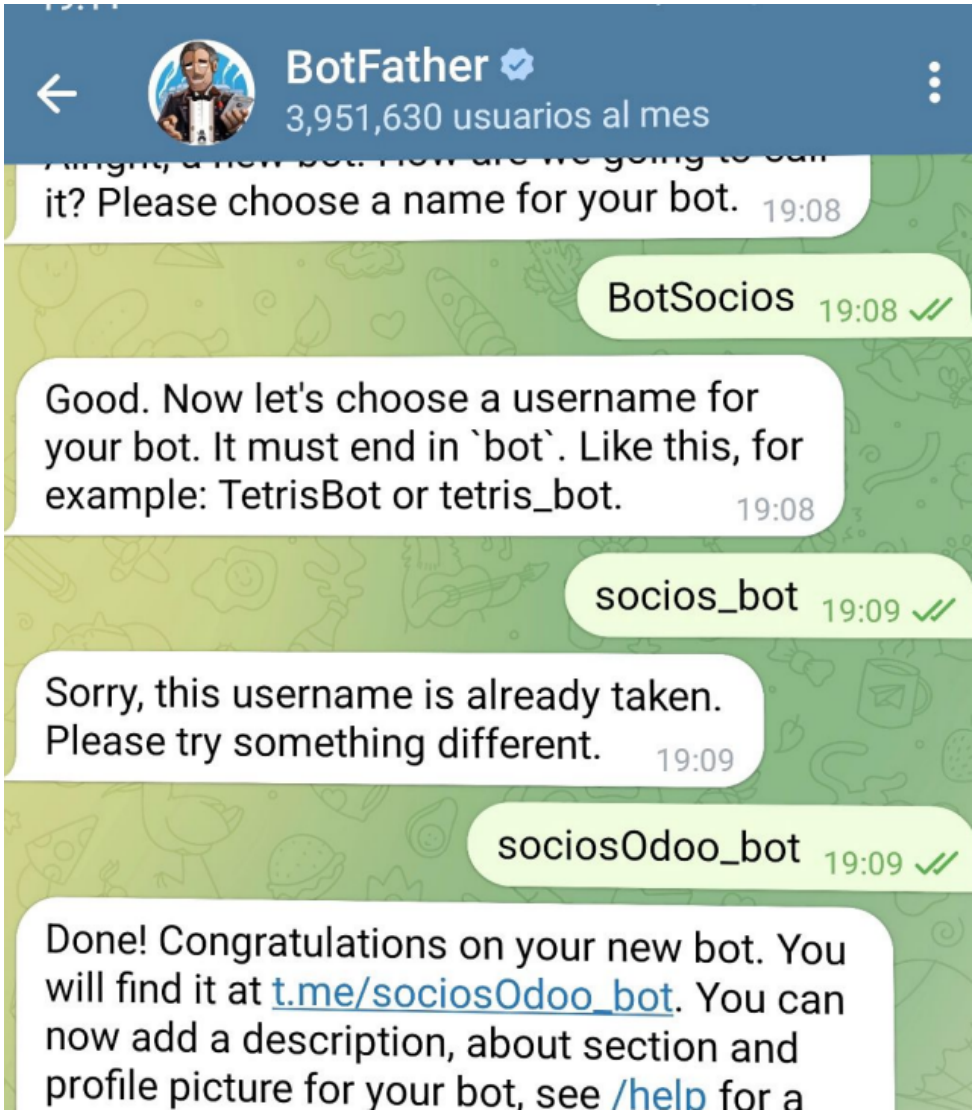

Gestión de liga
Equipos
Clasificacion
Partidos de la liga
Añadir equipo
Crear Nuevo Partido
My Company
A
Partidos de la liga
Buscar...
Medidas
Goles Casa
2,00
1,80
1,60
1,40
1,20
1,00
0,80
0,60
0,40
0,20
0,00
Barcelona
Real Madrid

2. Bot de Telegram conectado a la API REST

En esta actividad vamos a crear un bot de Telegram que se conecte a la API REST que creamos en la actividad anterior. Usaremos la librería 'python-telegram-bot' para crear el bot y la librería 'requests' para hacer las peticiones HTTP a la API REST. Dependiendo de la orden que se envíe se realizarán diferentes peticiones (GET, POST, PUT o DELETE) a la API, Odoo procesara esa solicitud y el bot devolverá el resultado al usuario.

2.1. Creación del bot

Primero de nada, debemos crear el propio bot en telegram. Para eso, abrimos Telegram y buscamos el bot 'BotFather'. Le enviamos el comando '/newbot' y seguimos las instrucciones para crear nuestro bot.



Nos dará un token que guardaremos y usaremos para conectar nuestro bot con la API de Telegram.

2.2. Creación y configuración de entorno para el bot

Ahora crearemos un entorno virtual para nuestro bot con el siguiente comando.

```
PS C:\Users\pmeibel\Desktop\laburo\SGE\ERP_Odoo\bot> python -m venv venv_bot
```

Accedemos al entorno virtual e instalaremos las librerías necesarias con pip(el python-telegram-bot y requests).

```
PS C:\Users\pmeibel\Desktop\laburo\SGE\ERP_Odoo\bot> .\venv_bot\Scripts\activate
(venv_bot) PS C:\Users\pmeibel\Desktop\laburo\SGE\ERP_Odoo\bot> pip install python-telegram-bot requests
```

2.3. Desarrollo

Posteriormente crearemos un archivo 'bot_socios.py' y añadiremos el código necesario para crear el bot. Aquí es donde deberemos usar por un lado el token que nos dio BotFather y por otro la URL de nuestra API REST.

```
TOKEN = "8248669096:AAF3Fc5YBrEakkj3MO8h92VcYyoKsRF3Z34"
API_URL = "http://localhost:8069/gestion/apirest/socio"
```

También deberemos indicar las funciones para interactuar con la API REST (crear, modificar, consultar y borrar socios).

```
def crear_socio(data):
    response = requests.post(API_URL, json=data)
    return response.json(), response.status_code

def modificar_socio(data):
    response = requests.put(API_URL, json=data)
    return response.json(), response.status_code

def consultar_socio(num_socio):
    payload = {"data": f'{{"num_socio": {num_socio}}}'}
    response = requests.get(API_URL, params=payload)
    return response.json(), response.status_code

def borrar_socio(num_socio):
    payload = {"data": f'{{"num_socio": {num_socio}}}'}
    response = requests.delete(API_URL, params=payload)
    return response.json(), response.status_code
```

Las funciones anteriores se usarán en la función principal del bot (handler), que se encargará de manejar los mensajes recibidos y llamar a las funciones correspondientes según el comando recibido.

```python
async def manejar_mensaje(update: Update, context: ContextTypes.DEFAULT_TYPE):
    texto = update.message.text.strip()
    chat_id = update.message.chat_id

    if texto.lower().startswith("crear"):
        try:
            params = texto[6:]
            data = {k.strip(): int(v) if k.strip() == "num_socio" else v.strip()
                    for k, v in (x.split("=") for x in params.split(","))}
            res, status = crear_socio(data)
            await context.bot.send_message(chat_id, f"Respuesta Crear ({status}): {res}")
        except Exception as e:
            await context.bot.send_message(chat_id, f"Error al crear: {e}")
        return

    elif texto.lower().startswith("modificar"):
        try:
            params = texto[10:]
            data = {k.strip(): int(v) if k.strip() == "num_socio" else v.strip()
                    for k, v in (x.split("=") for x in params.split(","))}
            res, status = modificar_socio(data)
            await context.bot.send_message(chat_id, f"Respuesta Modificar ({status}): {res}")
        except Exception as e:
            await context.bot.send_message(chat_id, f"Error al modificar: {e}")
        return

    elif texto.lower().startswith("consultar"):
        try:
            params = texto[9:]
            num_socio = int(params.split("=")[1])
            res, status = consultar_socio(num_socio)
            await context.bot.send_message(chat_id, f"Respuesta Consultar ({status}): {res}")
        except Exception as e:
            await context.bot.send_message(chat_id, f"Error al consultar: {e}")
        return

    elif texto.lower().startswith("borrar"):
        try:
            params = texto[7:]
            num_socio = int(params.split("=")[1])
            res, status = borrar_socio(num_socio)
            await context.bot.send_message(chat_id, f"Respuesta Borrar ({status}): {res}")
        except Exception as e:
            await context.bot.send_message(chat_id, f"Error al borrar: {e}")
        return

    else:
        await context.bot.send_message(chat_id, "Orden no soportada")
```

2.4. Arranque y funcionamiento

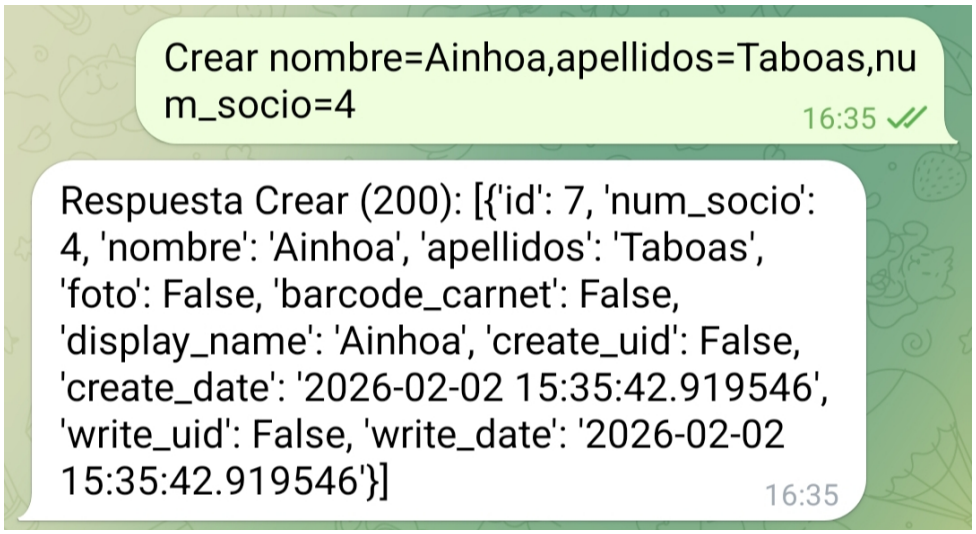
Con todo esto hecho podemos iniciar nuestro bot con el siguiente comando.

```
PS C:\Users\pmeibel\Desktop\laburo\SGE\ERP_Odoo\bot> .\venv_bot\Scripts\activate
(venv_bot) PS C:\Users\pmeibel\Desktop\laburo\SGE\ERP_Odoo\bot>
```

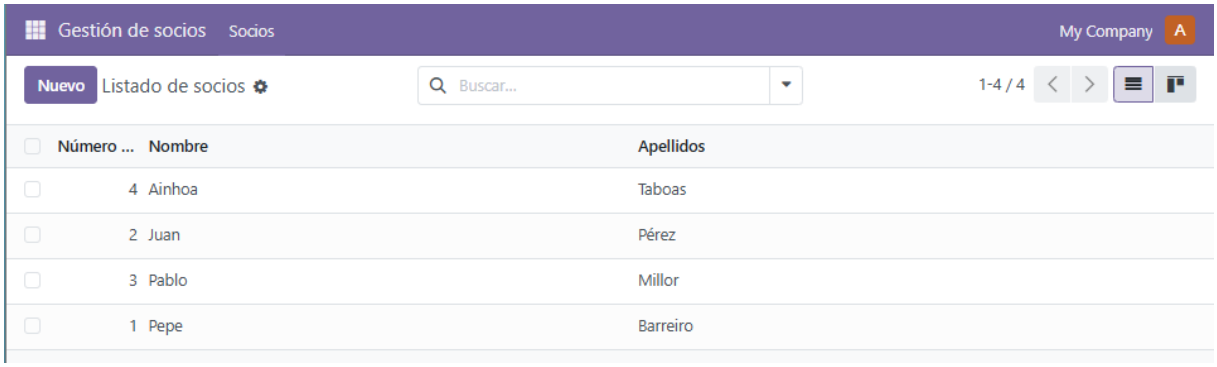
Desde dentro usaremos el comando 'python' junto con el nombre del archivo para iniciarlo y si todo esta correcto iniciará sin problema.

```
(venv_bot) PS C:\Users\pmeibel\Desktop\laburo\SGE\ERP_Odoo\bot> python .\bot_socios.py
Bot de socios iniciado...
```

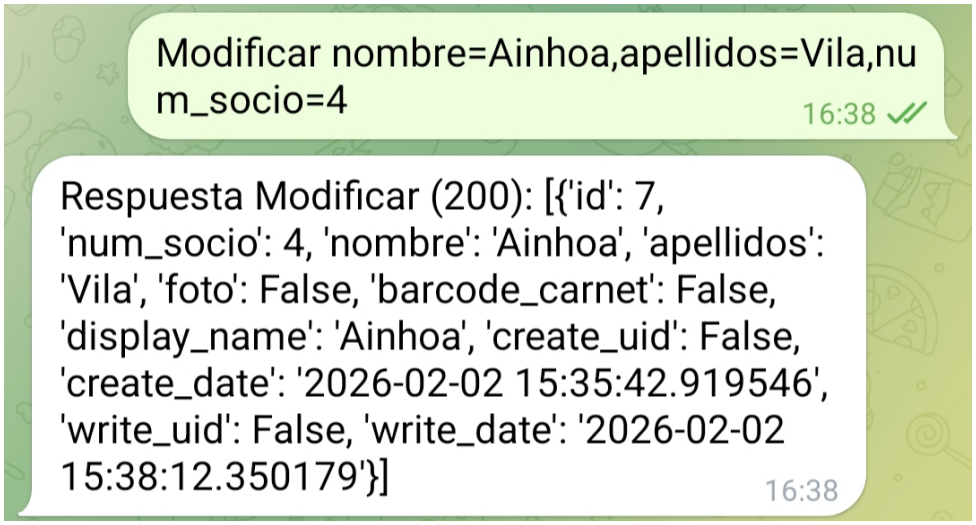
Ahora nos quedará probar los comandos uno por uno desde Telegram y ver si los cambios se reflejan en Odoo. Empecemos creando un socio.



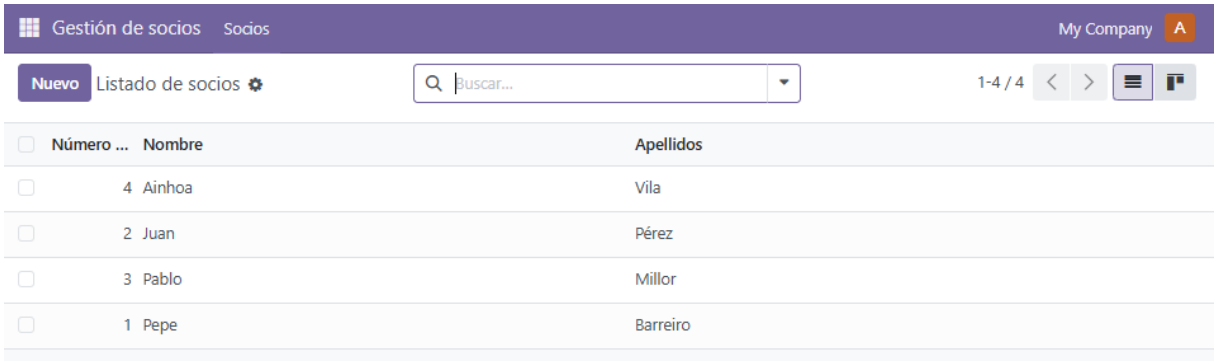
Y comprobamos en Odoo que se ha creado correctamente.



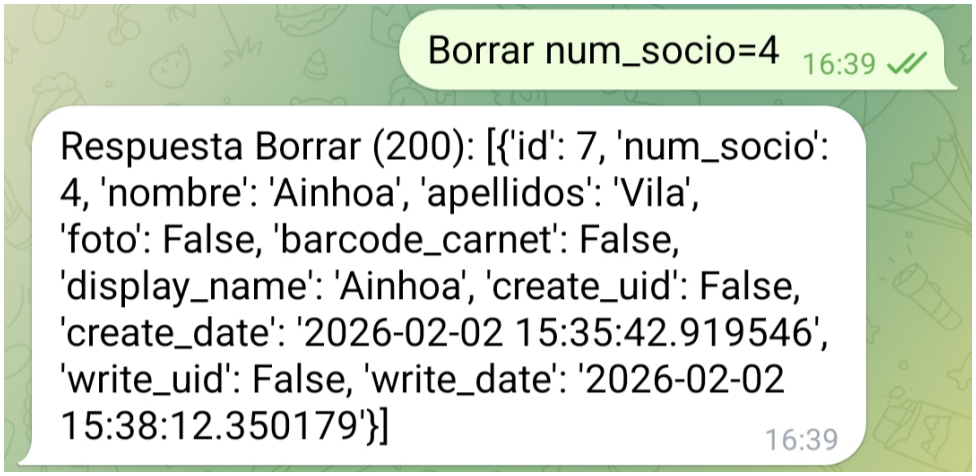
Probemos a modificarlo.



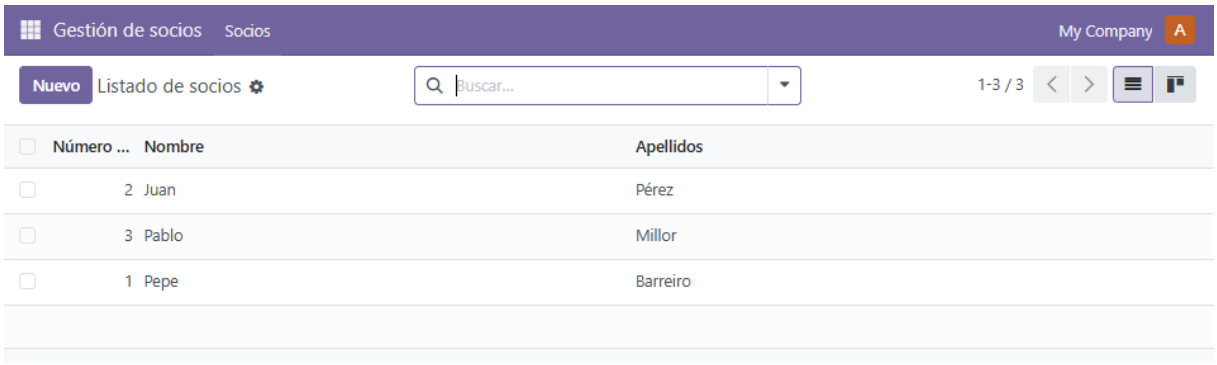
Revisamos en Odoo que el apellido ha sido modificado.



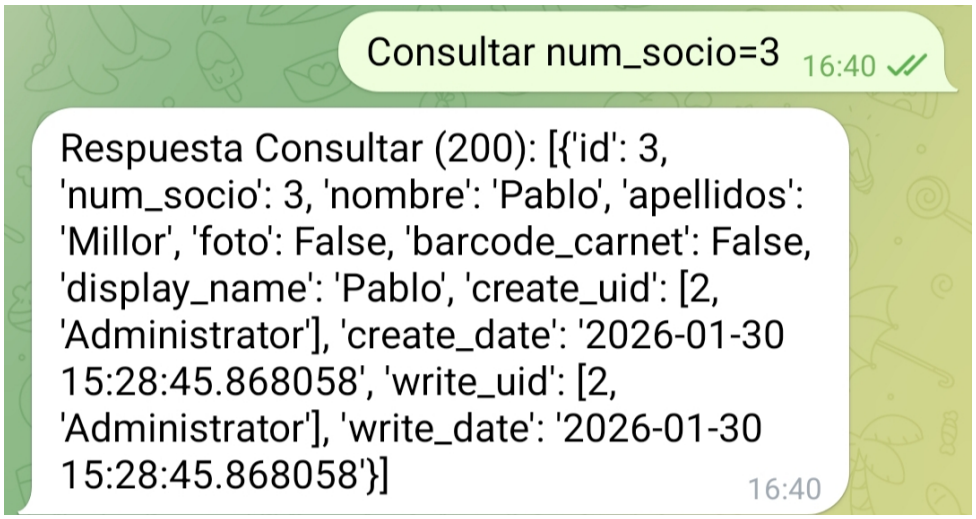
Ahora borraremos el socio que hemos creado.



Y comprobamos en Odoo que ha sido eliminado.



Para terminar las operaciones probaremos a consultar un socio (en este caso el que tenga el numero de socio 3).



Por último, comprobamos que al mandar cualquier otro mensaje el bot responde 'Orden no soportada'.

Si todo ha funcionado correctamente, nuestro bot estará terminado.

3. Generación de imágenes aleatorias con Web Controllers

Para esta parte de la práctica, vamos a crear un Web Controller que genere imágenes aleatorias tomando como base el módulo 'EJ09-GenerarBarcode' proporcionado.

3.1. Instalaciones necesarias

Primero deberemos acceder a nuestro contenedor de Odoo, para buscarlo usaremos el comando 'docker ps'.

```
PS C:\Users\pmeibel\Desktop\laburo\SGE\ERP_Odoo\Docker> docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED       STATUS         PORTS                                       NAMES
3379579c7956   odoo:18.0     "/usr/bin/odoo -c /e…"   6 weeks ago   Up 2 seconds   0.0.0.0:8069->8069/tcp, [::]:8069->8069/tcp   odoo18
a0c46bad56df   postgres:15   "docker-entrypoint.s…"   6 weeks ago   Up 3 seconds   0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp   odoo18_db
```

Esto nos mostrará los contenedores activos, buscaremos el ID del contenedor de Odoo y accederemos a el con el siguiente comando.

```
PS C:\Users\pmeibel\Desktop\laburo\SGE\ERP_Odoo\Docker> docker exec -u root -it 3379579c7956 bash
```

Una vez dentro usaremos el comando 'pip list' para ver las librerías instaladas.

```
root@3379579c7956:/# pip list
WARNING: Skipping /usr/lib/python3.12/dist-packages
Package             Version
------------------- -----------------
asn1crypto          1.5.1
attrs               23.2.0
Babel               2.10.3
beautifulsoup4      4.12.3
cached-property     1.5.2
cbor2               5.6.2
```

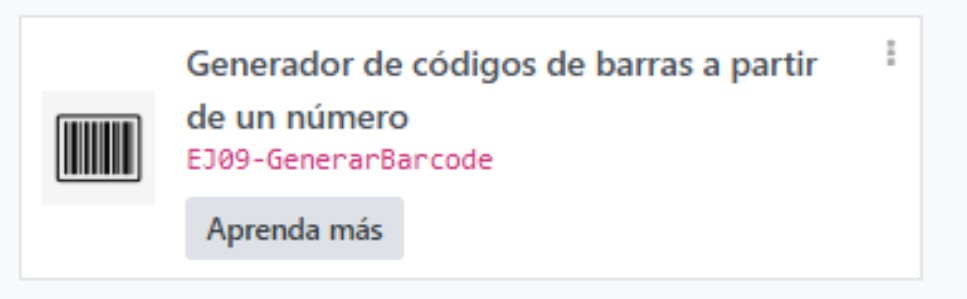
En cualquier caso, haremos un 'apt-get update' para actualizar lo necesario.

```
root@3379579c7956:/# apt-get update
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:3 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [34.8 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [1776 kB]
Get:5 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [1194 kB]
Get:6 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [3008 kB]
Get:7 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:8 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:9 http://archive.ubuntu.com/ubuntu noble/restricted amd64 Packages [117 kB]
```

Ahora instalaremos las librerías que vamos a necesitar con los siguiente comandos.

```
root@3379579c7956:/# apt-get install -y python3-pip python3-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Hecho esto, reiniciamos el contenedor para que los cambios tengan efecto. Y activamos el módulo en Odoo.



3.2. Prueba Web Controller de código de barras

Ahora podemos probar el Web Controller del código de barras que viene por defecto accediendo a la URL 'http://localhost:8069/generador/123456789012'.

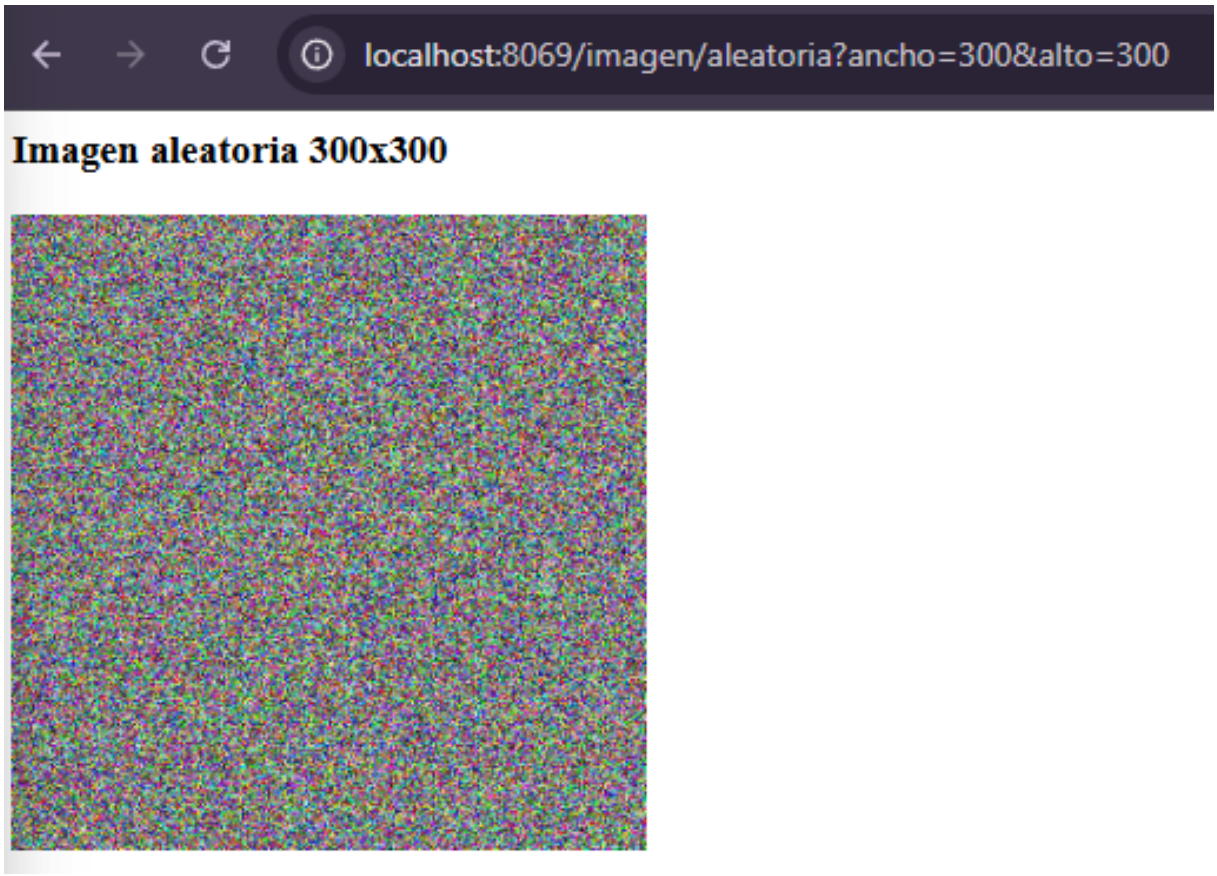
3.3. Creación de Web Controller para imágenes aleatorias

Vamos ahora entonces a añadir nuestro propio Web Controller para generar imágenes aleatorias. Añadiremos un archivo en mi caso 'generarimagen.py' y añadiremos el siguiente código, que generará pixeles de colores aleatorios con el alto y ancho que indiquemos.

La imagen se genera en formato PNG y se devolverá como respuesta al acceder a la URL.

```python
# -*- coding: utf-8 -*-
from odoo import http
from odoo.http import request
from PIL import Image
from io import BytesIO
import base64
import random

class GenerarImagenAleatoria(http.Controller):

    @http.route('/imagen/aleatoria', auth='public', cors='*', type='http')
    def crearImagen(self, ancho=200, alto=200, formato='html', **kw):
        # Convertimos parámetros a enteros
        try:
            ancho = int(ancho)
            alto = int(alto)
        except ValueError:
            return "Parámetros inválidos. Usa enteros para ancho y alto."

        # Creamos imagen RGB
        img = Image.new('RGB', (ancho, alto))

        # Generamos píxeles aleatorios
        pixels = [(random.randint(0,255), random.randint(0,255),
random.randint(0,255))
                  for _ in range(ancho * alto)]
        img.putdata(pixels)

        buffer = BytesIO()
        img.save(buffer, format='PNG')
        buffer.seek(0)

        if formato.lower() == 'png':
            # Devuelve la imagen directamente como PNG
            return request.make_response(
                buffer.getvalue(),
                headers=[('Content-Type', 'image/png')]
            )
        else:
            # Devuelve HTML con la imagen en Base64
            img_str = base64.b64encode(buffer.getvalue()).decode('utf-8')
            html = f'''
            <div>
                <h3>Imagen aleatoria {ancho}x{alto}</h3>
                <img src="{img_str}"/>
            </div>
            '''
            return html
```

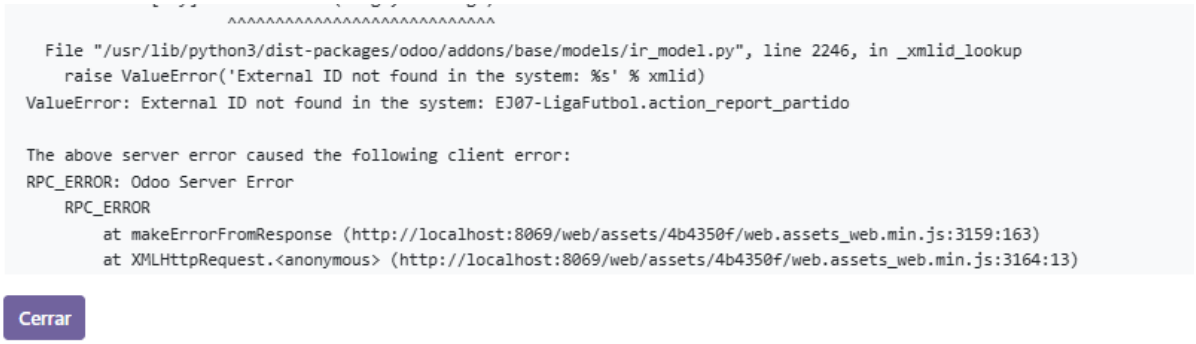
Con esto hecho, actualizamos el módulo y probamos el Web Controller accediendo a la URL 'http://localhost:8069/imagen/aleatoria?ancho=300&alto=300' (indicando el alto y ancho deseados).


localhost:8069/imagen/aleatoria?ancho=300&alto=300
Imagen aleatoria 300x300

4. Problemas encontrados

Durante la realización de esta práctica he tenido algunos problemas básicos como sintaxis de Python o permisos mal configurados.

*Pero el principal problema lo he tenido en el apartado 4 de la primera actividad, el informe PDF para los partidos. Al intentar generar el PDF, me da un error de 'External ID not found'.



Por la experiencia que tengo con Odoo y lo que he investigado, debería ser un problema con la referencia (que la hubiera escrito mal) o que no haya declarado el XML en el manifest y Odoo no pueda verlo, pero aún revisandolo varias veces no he conseguido encontrar el error, por lo que no he podido completar esa parte de la práctica.

Algunos problemas menores que he tenido también fueron con la creación del bot de Telegram, la recalculación de las puntuaciones de los partidos y los botones para alterar los goles, pero no espcialmente destacables

5. Conclusión

Esta práctica me ha servido para desenvolverme más en la creación y modificación de módulos en Odoo, la creación de Web Controllers, la interacción de una API REST con otro servicio diferente (en este caso, el bot de Telegram).