Your Paper

Birhan Fdez

4 de febrero de 2026

Índice

Volver al índice

1. Objetivo

El objetivo de este documento es la demostración del manejo sobre las vistas, wizards, informes y controladores web ademas de mostrar una prueba API REST mediante herramientas externas.

2. Actividad 01

En esta actividad se ha trabajado sobre el módulo base *EJ07-LigaFutbol*, realizando modificaciones tanto en los modelos como en las vistas, con el objetivo de ampliar la lógica de la liga y añadir nuevas funcionalidades solicitadas en el enunciado.

2.1. Reglas de puntuación especiales

En este apartado, se nos pide modificar el sistema de puntuación de los equipos para contemplar una regla especial. Esta regla no dice que cuando un equipo gana un partido con una diferencia de **4 o más goles**, además de los **3 puntos habituales por victoria**, obtiene **4 puntos adicionales**. Asimismo, el equipo perdedor en este tipo de partidos pierde **1 punto**.

Para implementar esta regla, nos dirigimos al apartado **liga_partido.py** del archivo **Models**. Una vez en el apartado, modificaremos la función **actualizoRegistrosEquipo** añadiendo los siguientes datos en las variables de los equipos para que obtengamos lo siguiente:

Viejas Variables De los Equipos

```
recordEquipo.victorias = 0
recordEquipo.empates = 0
recordEquipo.derrotas = 0
recordEquipo.goles_a_favor = 0
recordEquipo.goles_en_contra = 0
```

Nuevas Variables De los Equipos con los Puntos

```
recordEquipo.victorias = 0
recordEquipo.empates = 0
recordEquipo.derrotas = 0
recordEquipo.goles_a_favor = 0
recordEquipo.goles_en_contra = 0
recordEquipo.puntos = 0
```

Volver al índice

Con las variables modificadas, en el apartado de la lógica encargada de gestionar las victorias, empates y derrotas, añadiremos la lógica que se encargara de gestionar los puntos. Esto se realiza añadiendo el siguiente código:

Nuevas Lógica de Puntos para Equipo Local

```
for recordPartido in self.env['liga.partido'].search([]):
    if recordPartido.equipo_casa.nombre==recordEquipo.nombre:
        if recordPartido.goles_casa>recordPartido.goles_fuera:
            recordEquipo.victorias=recordEquipo.victorias+1
            if(recordPartido.goles_casa-recordPartido.goles_fuera)>=4:
                recordEquipo.puntos=recordEquipo.puntos+7
            else:
                recordEquipo.puntos=recordEquipo.puntos+3
        elif recordPartido.goles_casa<recordPartido.goles_fuera:
            recordEquipo.derrotas=recordEquipo.derrotas+1
            if(recordPartido.goles_fuera-recordPartido.goles_casa)>=4:
                recordEquipo.puntos=recordEquipo.puntos-1
        else:
            recordEquipo.empates=recordEquipo.empates+1
            #Añadimos un punto al equipo de casa
            recordEquipo.puntos=recordEquipo.puntos+1
```

Nuevas Lógica de Puntos para Equipo Visitante

```
    if recordPartido.equipo_fuera.nombre==recordEquipo.nombre:
        if recordPartido.goles_casa<recordPartido.goles_fuera:
            recordEquipo.victorias=recordEquipo.victorias+1
            if (recordPartido.goles_fuera-recordPartido.goles_casa)>=4:
                recordEquipo.puntos=recordEquipo.puntos+7
            else:
                recordEquipo.puntos=recordEquipo.puntos+3
        elif recordPartido.goles_casa>recordPartido.goles_fuera:
            recordEquipo.derrotas=recordEquipo.derrotas+1
            if(recordPartido.goles_casa-recordPartido.goles_fuera)>=4:
                recordEquipo.puntos=recordEquipo.puntos-1
        else:
            recordEquipo.empates=recordEquipo.empates+1
            recordEquipo.puntos=recordEquipo.puntos+1
```

Volver al índice

```python
def actualizoRegistrosEquipo(self):
    #Recorremos partidos y equipos
    for recordEquipo in self.env['liga.equipo'].search([]):
        #Como recalculamos todo, ponemos de cada equipo todo a cero
        recordEquipo.victorias=0
        recordEquipo.empates=0
        recordEquipo.derrotas=0
        recordEquipo.goles_a_favor=0
        recordEquipo.goles_en_contra=0
        recordEquipo.puntos=0

        for recordPartido in self.env['liga.partido'].search([]):
            #Si es el equipo de casa
            if recordPartido.equipo_casa.nombre==recordEquipo.nombre:
                #Si los goles del equipo de casa es mayor que el visitante
                if recordPartido.goles_casa>recordPartido.goles_fuera:
                    #Añadimos una victoria al equpo de casa
                    recordEquipo.victorias=recordEquipo.victorias+1
                    #Si la diferencia de los goles del equipo de casa es >=4
                    if(recordPartido.goles_casa-recordPartido.goles_fuera)>=4:
                        #Añadimos 7 puntos al equipo de casa
                        recordEquipo.puntos=recordEquipo.puntos+7
                    else:
                        #Añadimos 3 puntos al equipo de casa
                        recordEquipo.puntos=recordEquipo.puntos+3
                #Si los goles del equipo de casa es menor que el visitante
                elif recordPartido.goles_casa<recordPartido.goles_fuera:
                    #Añadimos una derrota al equipo de casa
                    recordEquipo.derrotas=recordEquipo.derrotas+1
                    #Si la diferencia de los goles del equipo visitante es >=4
                    if(recordPartido.goles_fuera-recordPartido.goles_casa)>=4:
                        #Añadimos 7 puntos al equipo de casa
                        recordEquipo.puntos=recordEquipo.puntos-1
                #Si los goloes de ambos equipos son iguales
                else:
                    #Añadimos un empate al equipo de casa
                    recordEquipo.empates=recordEquipo.empates+1
                    #Añadimos un punto al equipo de casa
                    recordEquipo.puntos=recordEquipo.puntos+1

                #Sumamos goles a favor y en contra
                recordEquipo.goles_a_favor=recordEquipo.goles_a_favor+recordPartido.goles_casa
                recordEquipo.goles_en_contra=recordEquipo.goles_en_contra+recordPartido.goles_fuera
```

Con este nuevo código añadido, nos iremos al apartado **liga_equipo.py** para realizar un cambio en el atributo de los equipos. Esta modificación se hace eliminando la vieja lógica y solo declarando el atributo puntos.

Vieja Lógica de Puntos

```
puntos= fields.Integer( compute="_compute_puntos",default=0, store=True)

@api.depends('victorias','empates')
def _compute_puntos(self):
    for record in self:
        record.puntos = record.victorias * 3 + record.empates
```

Nuevo Atributo de Puntos

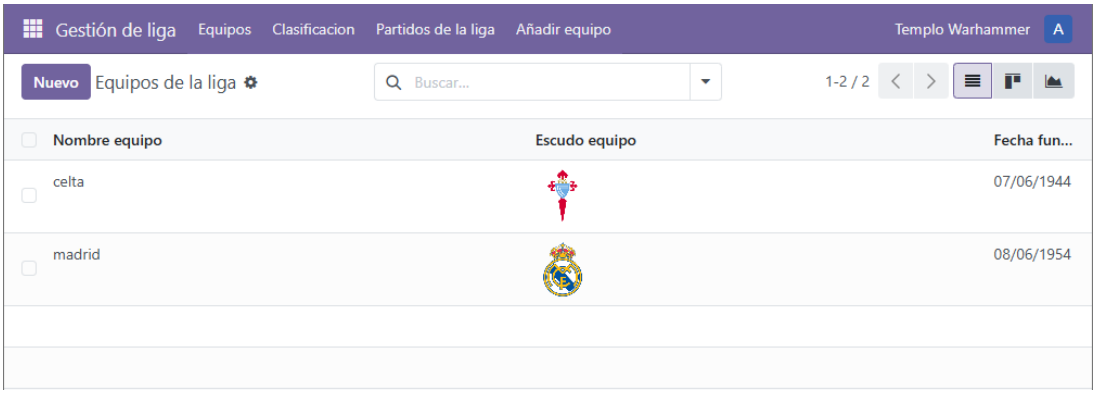
```
    puntos= fields.Integer(default=0)
```

Volver al índice

Esto nos tiene que quedar de la siguiente forma:

```python
#Partidos jugados, ganados, empatados, perdidos
victorias=fields.Integer(default=0)
empates=fields.Integer(default=0)
derrotas=fields.Integer(default=0)

jugados= fields.Integer( compute="_compute_jugados", store=True)

@api.depends('victorias','empates','derrotas')
def _compute_jugados(self):
    for record in self:
        record.jugados = record.victorias + record.empates + record.derrotas

puntos=fields.Integer(default=0)
"""
puntos= fields.Integer( compute="_compute_puntos",default=0, store=True)

@api.depends('victorias','empates')
def _compute_puntos(self):
    for record in self:
        record.puntos = record.victorias * 3 + record.empates
"""
```

Con la nueva lógica implementada, nos dirigiremos al módulo de **Odoo** donde crearemos dos equipos que se enfrentaran para comprobar su correcto funcionamiento.



Una vez comprobado su correcto funcionamiento, podremos avanzar al siguiente apartado.

2.2. Botones para modificar los goles de los partidos

Esta sección de la activada, nos pide que realicemos una actualización en la vistas **Kanban y Lista** del modelo **liga.partido**

La actualización consiste en añadir dos botones que permitan modificar de forma general los goles registrados en cada uno de los partidos que exista. Cada botón contara con la función **de sumar 2 goles al equipo local o al equipo visitante**.

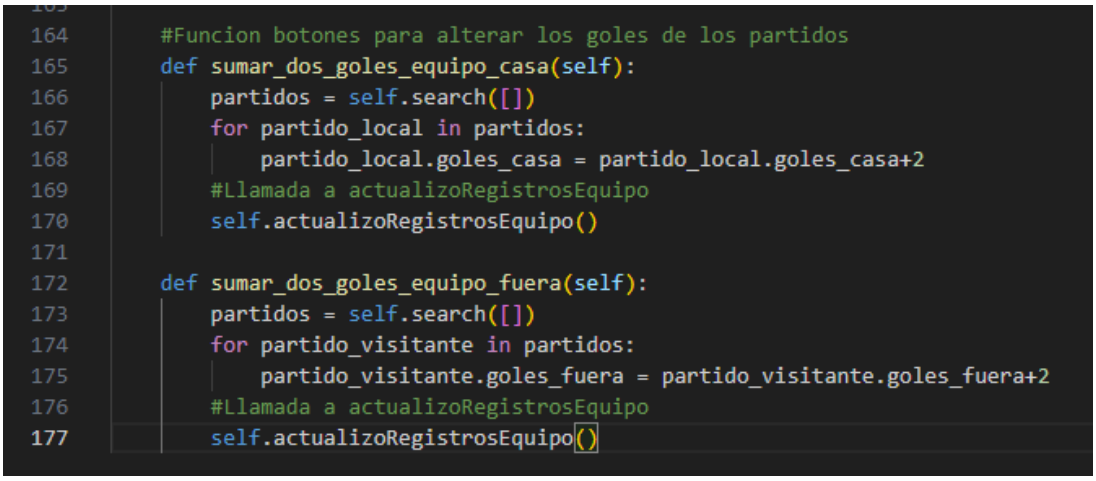
Para poder realizar esta actualización, nos dirigiremos al **liga_partido.py**. Una vez entremos, nos dirigiremos al final del código para agregar las siguientes dos funciones que manejaran a cada uno de los botones:

Lógica del Botón Para Sumar Goles al Equipo Local

```
def sumar_dos_goles_equipo_casa(self):
    partidos = self.search([])
    for partido_local in partidos:
        partido_local.goles_casa = partido_local.goles_casa+2
    #Llamada a actualizoRegistrosEquipo
    self.actualizoRegistrosEquipo()
```

Lógica del Botón Para Sumar Goles al Equipo Visitante

```
def sumar_dos_goles_equipo_fuera(self):
    partidos = self.search([])
    for partido_visitante in partidos:
        partido_visitante.goles_fuera = partido_visitante.goles_fuera+2
    #Llamada a actualizoRegistrosEquipo
    self.actualizoRegistrosEquipo()
```



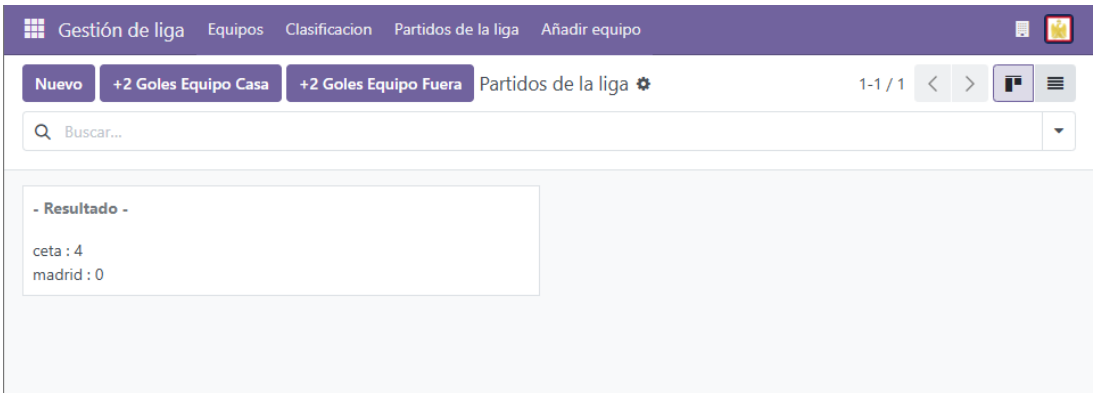
Tras añadir esta dos nuevas funciones, nos dirigiremos al apartado de **liga_partido.xml** en la carpeta de vistas del modulo. Dentro de la vista, se añadirá el siguiente código a las vistas de **Kanban y Lista**:

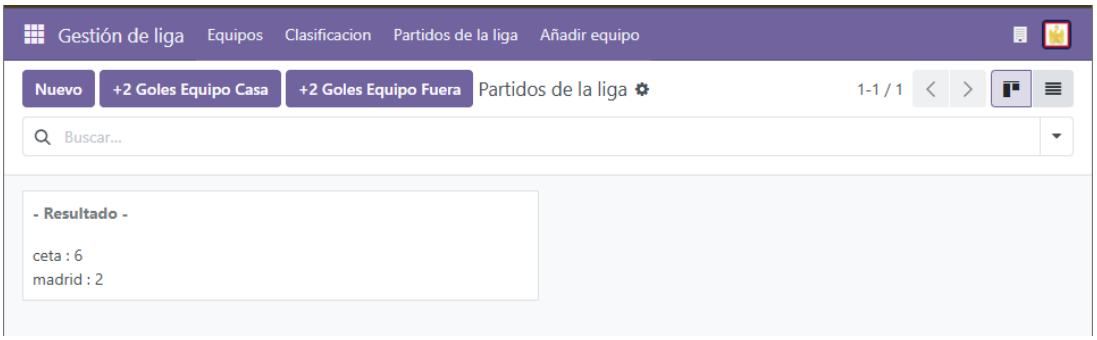
Lógica del Botón Para Sumar Goles al Equipo Visitante

```
<!-- Botones para sumar goles -->
<header>
    <!-- Botones para sumar goles al equipo local -->
    <button name="sumar_dos_goles_equipo_casa" type="object" string="+2 Goles Equipo
    <!-- Botones para sumar goles al equipo visitante -->
    <button name="sumar_dos_goles_equipo_fuera" type="object" string="+2 Goles Equip
</header>
```

```
<!-- Vista Tree -->
<record id="liga_partido_view_list" model="ir.ui.view">
    <field name="name">Lista de partidos de la liga</field>
    <field name="model">liga.partido</field>
    <field name="arch" type="xml">
        <list>
            <header>
                <button name="sumar_dos_goles_equipo_casa" type="object" string="+2 Goles Equipo Casa" display="always" class="oe_highlight"/>
                <button name="sumar_dos_goles_equipo_fuera" type="object" string="+2 Goles Equipo Fuera" display="always" class="oe_highlight"/>
            </header>
            <!-- Indicamos que atributos usaremos al hacer la vista Tree -->
            <field name="equipo_casa" />
            <field name="goles_casa" />
            <field name="equipo_fuera" />
            <field name="goles_fuera" />

        </list>
    </field>
</record>


<!-- Vista Kanban -->
<record id="liga_partido_view_kanban" model="ir.ui.view">
    <field name="name">Lista de partidos de la liga</field>
    <field name="model">liga.partido</field>
    <field name="arch" type="xml">
        <!-- Agrupamos por el atributo "parent_id"-->
        <kanban>
            <header>
                <button name="sumar_dos_goles_equipo_casa" type="object" string="+2 Goles Equipo Casa" display="always" class="oe_highlight"/>
                <button name="sumar_dos_goles_equipo_fuera" type="object" string="+2 Goles Equipo Fuera" display="always" class="oe_highlight"/>
            </header>
            <!-- Indicamos que atributos usaremos al hacer la vista Kanban -->

            <field name="equipo_casa" />
```

Con las modificaciones a las vistas realizadas, nos dirigiremos a modulo desde **Odoo** para actualizarlo y comprobar su correcto funcionamiento.
Primero realizaremos una prueba desde la vista **Kanban**





Para terminar, comprobaremos la vista **Lista**

Con la confirmación del correcto funcionamiento de los botones en ambas vistas, daremos completada esta apartado.

2.3. Web controller para eliminar empates

Continuando con los ejercicios de la **Actividad 1**, en esta sección se nos pide que realicemos la creación de un **Web Controller** cuyo trabajo **sera eliminar los partido que estén empatado** para luego devolver un mensaje indicando el número que se eliminó.

Para comenzar a trabajar, ingresaremos al archivo **Manin.py** de la carpeta **controller** del modulo. Una vez que ingresemos, dentro de la clase **class Main(http.Controller):** crearemos el siguiente código:

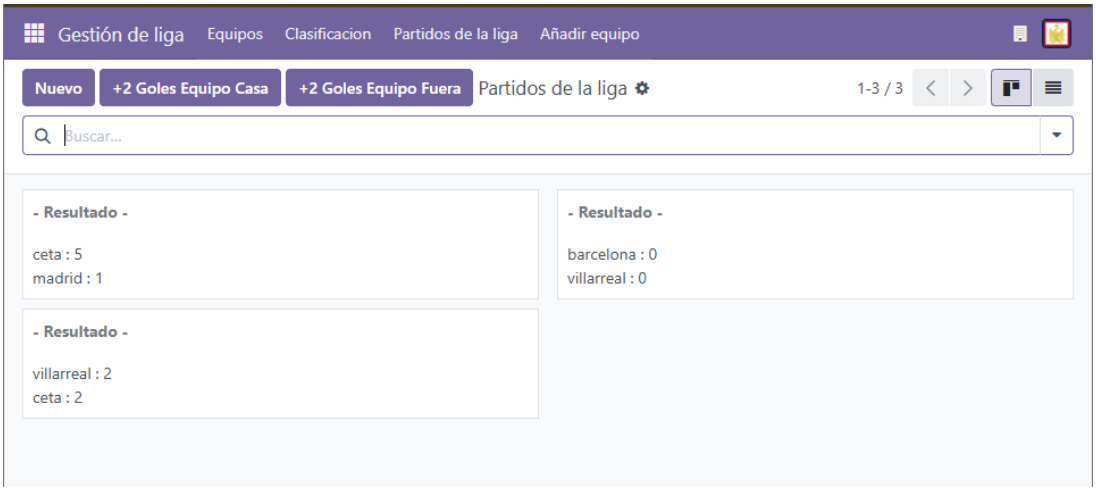
Código de la Función del Web controller

```
#URL del Web controller http://localhost:9001/ligafutbol/eliminarempates
@http.route('/ligafutbol/eliminarempates', type='http', auth='none')
def funcion_eliminar_empates(self):
    try:
        # Buscamos todos los partidos
        lista_partidos = request.env['liga.partido'].sudo().search([])
        #Lista de los partidos empatados
        lista_partidos_empatados=[]

        # Filtramos los partidos que terminaron en empate
        for partido in lista_partidos:
            if partido.goles_casa == partido.goles_fuera:
                lista_partidos_empatados.append(partido)
            # Contamos los partidos empatados
            num_partidos_empatados=len(lista_partidos_empatados)

        #Comprobamos si existe algun partido empatado
        if num_partidos_empatados>0:
            for partido_empatado in lista_partidos_empatados:
                # Eliminamos los partidos empatados
                partido_empatado.unlink()
            mensaje=json.dumps({
        'mensaje': f'Se eliminaron {num_partidos_empatados} partidos empatado'
            })
        else:
            mensaje=json.dumps({
                'mensaje': f'No existen partidos empatado para eliminar'
            })
        # Retornamos JSON con la cantidad eliminada
        return mensaje
    except Exception as e:
                return json.dumps({"error": str(e)})
```

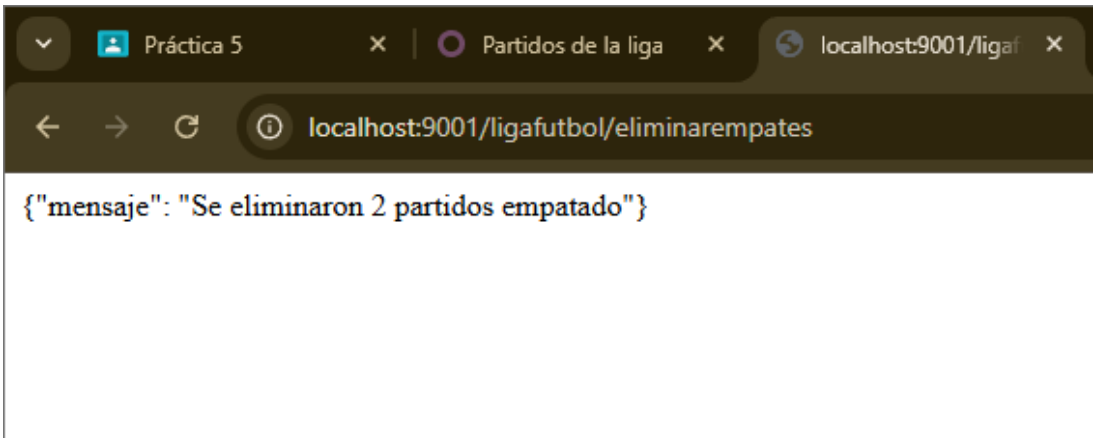
De tal forma que nos deberá de quedar así en nuestro archivo:

```python
#Se puede probar accediendo a http://localhost:9001/ligafutbol/eliminarempates
@http.route('/ligafutbol/eliminarempates', type='http', auth='none')
def funcion_eliminar_empates(self):
    try:
        # Buscamos todos los partidos
        lista_partidos = request.env['liga.partido'].sudo().search([])

        #Lista de los partidos empatados
        lista_partidos_empatados=[]

        # Filtramos los partidos que terminaron en empate
        for partido in lista_partidos:

            if partido.goles_casa == partido.goles_fuera:
                lista_partidos_empatados.append(partido)

            # Contamos los partidos empatados
            num_partidos_empatados=len(lista_partidos_empatados)

        #Comprobamos si existe algun partido empatado
        if num_partidos_empatados>0:

            for partido_empatado in lista_partidos_empatados:
                # Eliminamos los partidos empatados
                partido_empatado.unlink()

            mensaje=json.dumps({'mensaje': f'Se eliminaron {num_partidos_empatados} partidos empatado'})
        else:
            mensaje=json.dumps({'mensaje': f'No existen partidos empatado para eliminar'})

        # Retornamos JSON con la cantidad eliminada
        return mensaje

    except Exception as e:
            return json.dumps({"error": str(e)})
```

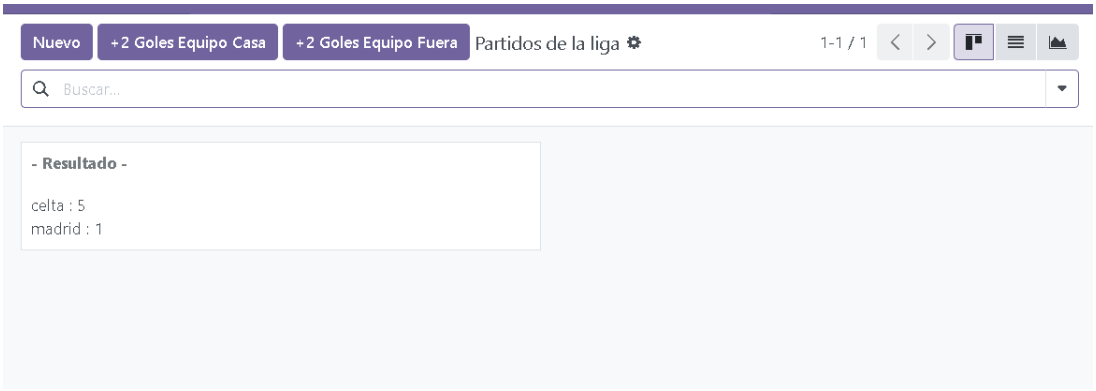
Con el código implementado, realizaremos una prueba para comprobar su correcto funcionamiento.
Lo primero que haremos, es crear dos partido cuyo resultado sera un empate:



Una vez creados, accederemos a la url para borrar los nuevos partidos creados.
http://localhost:9001/ligafutbol/eliminarempates



Como se puede observar en la imagen, el **Web Controller** elimino los dos partidos empatados y nos informo con un mensaje. Para confirmar del todo la eliminación de los partidos, regresaremos a la vista y comprobaremos que no estén.

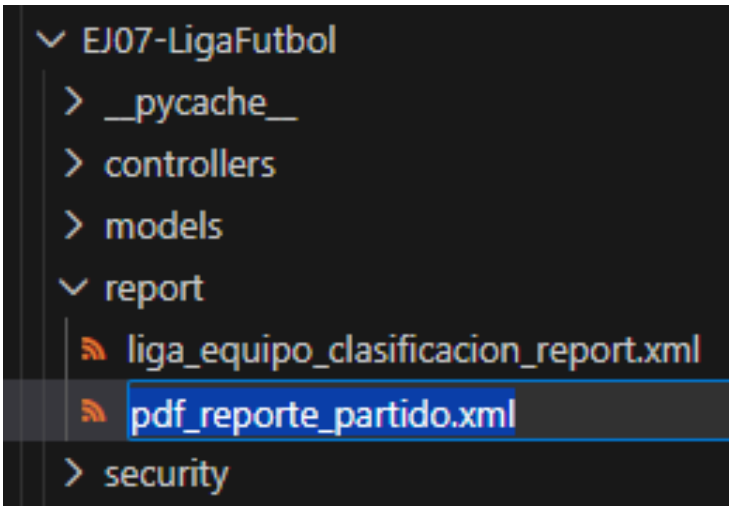


Al ingresar, observamos que la eliminación ha sido un éxito, confirmando el correcto funcionamiento y dando por finalizado el ejercicio.

2.4. Informe PDF por partido

Para realizar este ejercicio que nos pide crear **un informe PDF de cada partido**, tendremos que crear un **Report**.

Para ello, nos dirigimos a **Models/Reports**.Dentro de esta carpeta, crearemos un nuevo archivo que se llamará **pdf_reporte_partido.xml**



Una vez creado el archivo, antes de comenzar, nos dirigiremos al apartado de **'data'** en el archivo **___manifest___.py**.Dentro, añadiremos el archivo para que el modulo pueda reconocerlo:

```
#Aqui distintas vistas de equipo (vistas diferentes, mismo modelo)
'views/liga_equipo.xml',
'views/liga_equipo_clasificacion.xml',
#Vista a un informe
'report/liga_equipo_clasificacion_report.xml',
#Plantilla Reporte PDF Partido
'report/pdf_reporte_partido.xml',
#Aqui vista sobre los partidos
'views/liga_partido.xml',
```

Con la localización del archivo añadido, lo siguiente que realizaremos en implementar la función del reporte. Los primero que realizaremos es diseñar la apariencia que tendrá la plantilla:

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

    <!-- Plantilla principal del PDF -->
    <template id="plantilla_pdf_reporte_partido">
        <t t-call="web.html_container">
            <t t-call="web.external_layout">

                <!-- Estilos CSS -->
                <style>
                    h3 {
                        text-align: center;
                        font-size: 30px;
                        color: #0c0c0c;
                        margin-bottom: 15px;
                        font-family: Arial, sans-serif;
                    }

                    table {…
                    }

                    table th, table td {…
                    }

                    table th {…
                    }

                    .section {…
                    }
                </style>

                <!-- Header -->
                <div class="header">
                    <h1>Informe del Partido</h1>
                </div>

                <!-- Contenido principal -->
                <div class="section">
                    <t t-call="EJ07-LigaFutbol.contenido_partido"/>
                </div>

            </t>
        </t>
    </template>
```

Esto seguido de los datos que contendrá:

```
<!-- Template de contenido del partido -->
<template id="contenido_partido">
    <t t-if="docs">
        <t t-set="partido" t-value="docs and docs[0]"/>
        <table>
            <tr>
                <th>Equipo Local</th>
                <td><span t-field="partido.equipo_casa.nombre"/></td>
            </tr>
            <tr>
                <th>Goles Local</th>
                <td><span t-field="partido.goles_casa"/></td>
            </tr>
            <tr>
                <th>Equipo Visitante</th>
                <td><span t-field="partido.equipo_fuera.nombre"/></td>
            </tr>
            <tr>
                <th>Goles Visitante</th>
                <td><span t-field="partido.goles_fuera"/></td>
            </tr>
            <tr>
                <th>Resultado</th>
                <td>
                    <t t-if="partido.goles_casa &gt; partido.goles_fuera">
                        Victoria del local
                    </t>
                    <t t-elif="partido.goles_casa &lt; partido.goles_fuera">
                        Victoria del visitante
                    </t>
                    <t t-else="">
                        Empate
                    </t>
                </td>
            </tr>
            <tr>
                <th>Diferencia de Goles</th>
                <td>
                    <span t-esc="abs(partido.goles_casa - partido.goles_fuera)"/>
                </td>
            </tr>
        </table>
    </t>
    <t t-else="">
        <p>No hay partidos para mostrar.</p>
    </t>
</template>
```

Por ultimo, añadiremos la acción del reporte:

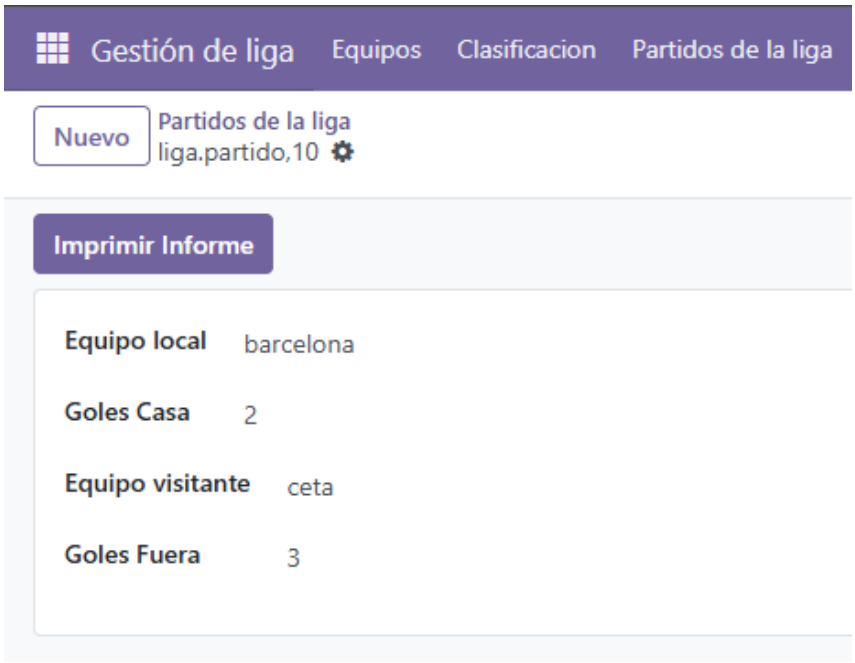
```
<!-- Acción del reporte PDF -->
<record id="action_report_partido_pdf" model="ir.actions.report">
    <field name="name">Informe del Partido</field>
    <field name="model">liga.partido</field>
    <field name="report_type">qweb-pdf</field>
    <field name="report_name">EJ07-LigaFutbol.plantilla_pdf_reporte_partido</field>
    <field name="print_report_name">
        'Partido - %s vs %s' % (object.equipo_casa.nombre, object.equipo_fuera.nombre)
    </field>
</record>
```

Con el archivo, finalizado, nos dirigiremos al archivo de vista **liga_partido.xml** para añadir el botón de descarga del PDF en la cabecera del partido deseado desde la vista de **Formulario**:

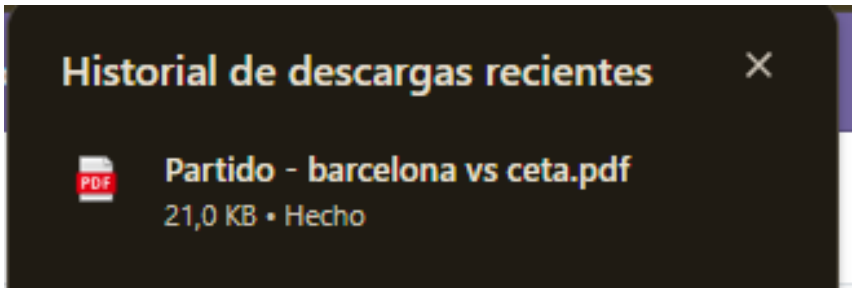
Botón Informe PDF

```
<header>
    <!-- Boton para descargar el reporte del partido-->
    <button
        name="%(EJ07-LigaFutbol.action_report_partido_pdf)d"
        string="Imprimir Informe"
        type="action"
        class="btn-primary"/>
</header>
```

Por último nos dirigiremos a los partidos, para crear un partido nuevo y comprobar su correcto funcionamiento.



Como se puede observar en la imagen, la creación fue un éxito.
Para dar por concluido con el generador de informes, realizaremos una impresión:

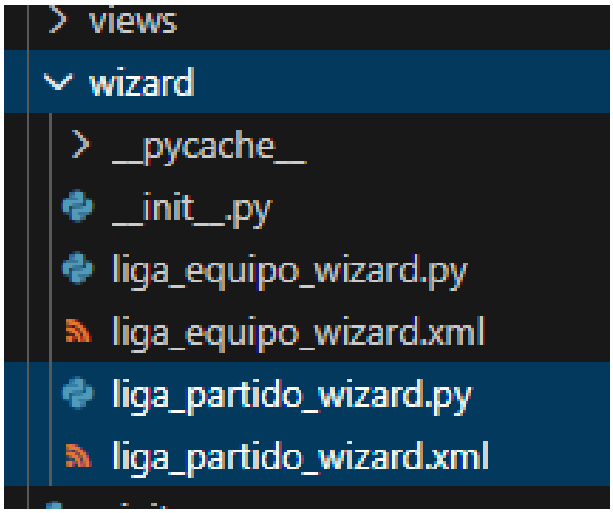


Siendo el resultado:

Equipo Local	celta
Goles Local	5
Equipo Visitante	madrid
Goles Visitante	1
Resultado	Victoria del local
Diferencia de Goles	4

2.5. Wizard para nuevos partidos

En este apartado, desarrollaremos un Wizard que permitirá al usuario crear nuevos partidos de forma mas rápida y sencilla, ya que se podrá ingresar toada la información con una sola ventana emergente.
Lo primero que realizaremos es crear un par de archivos nuevos que contendrán la función y vista de esta nueva verata emergente.
Estos los crearemos dentro de la carpeta **Wizard**



Con los archivos creados, añadiremos la la ruta del archivo de vista del **Wizard** en el apratado de **'Data'** deñ archivo **___manifest___.py**.

```
#Aqui distintas vistas de equipo (vistas diferentes, mismo modelo)
'views/liga_equipo.xml',
'views/liga_equipo_clasificacion.xml',
#Vista a un informe
'report/liga_equipo_clasificacion_report.xml',
#Plantilla Reporte PDF Partido
'report/pdf_reporte_partido.xml',
#Aqui vista sobre los partidos
'views/liga_partido.xml',
#Añadimos un Wizard para introducir equipos
'wizard/liga_equipo_wizard.xml',
#Añadimos un Wizard para introducir partidos
'wizard/liga_partido_wizard.xml'
```

Con la ruta del archivo asignada, realizaremos una modificación final en el archivo **___init___.py** donde añadiremos el modelo.

Practica1 > Practica_Docker > addons > EJ07-LigaFutbol > wizard > __init__.py

```
1  # -*- coding: utf-8 -*-
2  from . import liga_equipo_wizard
3  from . import liga_partido_wizard
```

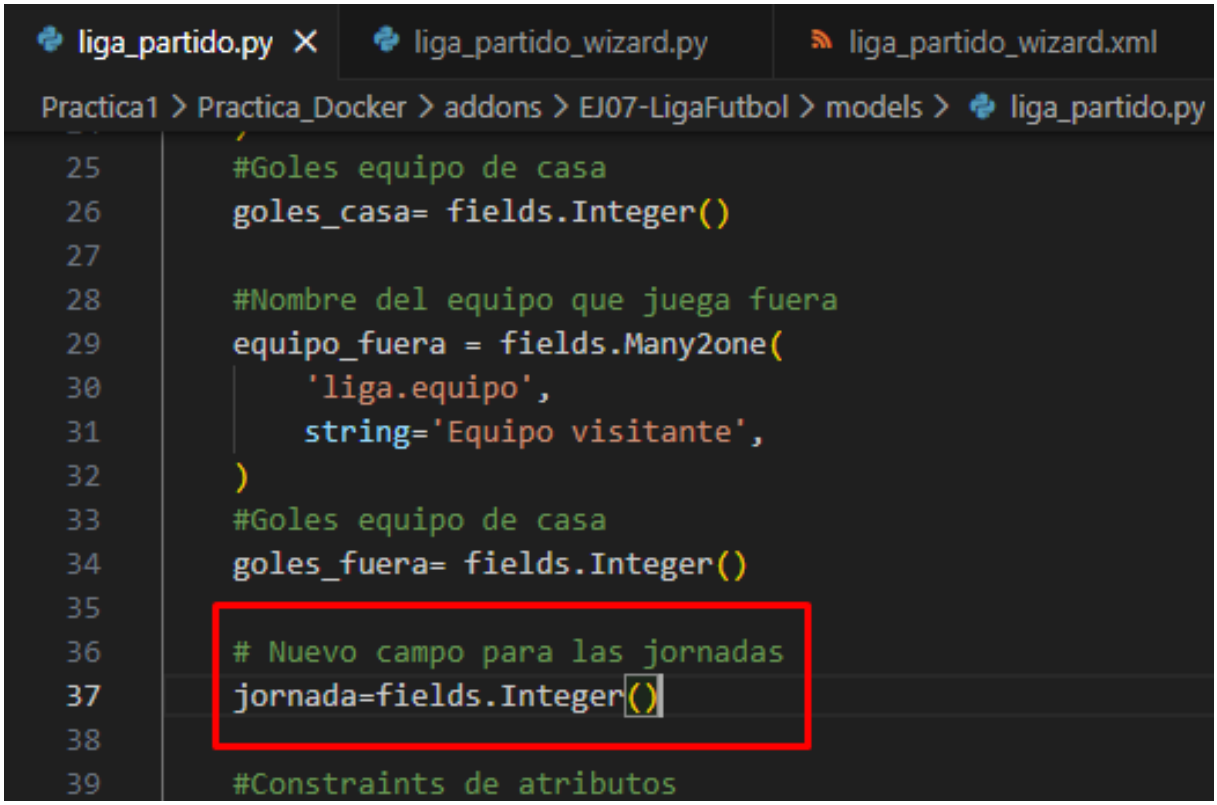
Con los pasos previos finalizado, nos dirigimos al archivo **liga_partido_wizard.py** para codifiar lo siguiente:

```python
# -*- coding: utf-8 -*-
from odoo import models, fields

class LigaPartidoWizard(models.TransientModel):
    #Nombre y Descripción
    _name='liga.partido.wizard'
    _descripcion='Wizard para crear partidos de liga'

    # Campos del modelo que usaremos en el Wizard
    equipo_casa=fields.Many2one('liga.equipo', string='Equipo de Casa', required=True)
    equipo_fuera=fields.Many2one('liga.equipo', string='Equipo Visitante', required=True)

    goles_casa=fields.Integer(string='Goles Equipo de Casa', default=0)
    goles_fuera=fields.Integer(string='Goles Equipo Visitante', default=0)

    # Nuevo campo para las jornadas
    jornada=fields.Integer(string='Jornada', required=False, default=1)

    # Funcion para que se llame desde el Wizard
    def add_liga_partido(self):
        # Obtenemos referencia al modelo destino
        ligaPartidoModel = self.env['liga.partido']
        # Recorremos porque self referencia a todo el modelo
        for wiz in self:
            # Creamos un registro en "liga.partido"
            ligaPartidoModel.create({
                'equipo_casa': wiz.equipo_casa,
                'equipo_fuera': wiz.equipo_fuera,
                'goles_casa': wiz.goles_casa,
                'goles_fuera': wiz.goles_fuera,
                'jornada': wiz.jornada,
            })
```

Una vez finalicemos con la función, en el archivo **liga_partido_wizard.xml** ingresaremos el siguinte código

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <!-- Vista del wizard para introducir un partido -->
    <record id='liga_partido_wizard_form' model='ir.ui.view'>
        <field name='name'>Wizard para introducir un Partido</field>
        <field name='model'>liga.partido.wizard</field>
        <field name='arch' type='xml'>
            <form string="Introducir datos de un partido">
                <sheet>
                    <group>
                        <field name='equipo_casa'/>
                        <field name='equipo_fuera'/>
                    </group>
                    <group>
                        <field name='goles_casa'/>
                        <field name='goles_fuera'/>
                        <field name='jornada'/>
                    </group>
                </sheet>
                <footer>
                    <button string='Añadir' name='add_liga_partido' class='btn-primary' type='object'/>
                    <button string='Cancel' class='btn-default' special='cancel'/>
                </footer>
            </form>
        </field>
    </record>
    <!-- Acción para abrir el wizard de partido -->
    <record id="action_wizard_liga_partido" model="ir.actions.act_window">
        <field name="name">Añadir partido</field>
        <field name="res_model">liga.partido.wizard</field>
        <field name="view_mode">form</field>
        <field name="target">new</field>
    </record>
    <!-- Menú para acceder al wizard de partido -->
    <menuitem id="menu_wizard_liga_partido" parent="liga_base_menu" action="action_wizard_liga_partido" sequence="20" />
</odoo>
```

Finalmente, nos dirigimos al archivo de **liga__partido.py** para crear un nuevo campo encargado de gestionar las jornadas de los partidos.



Con el nuevo campo agregado, nos dirigimos al archivo **security/ir.model.access.csv** para agregar lo siguiente:

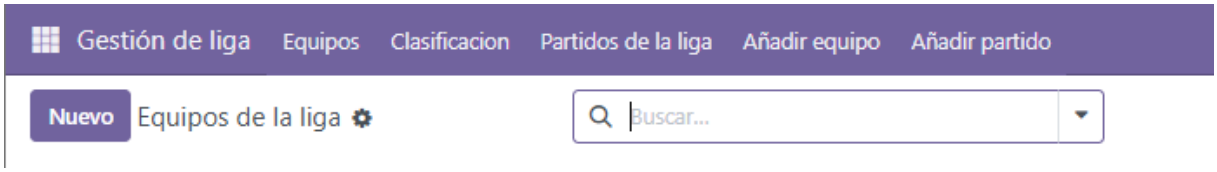
Datos del CSV

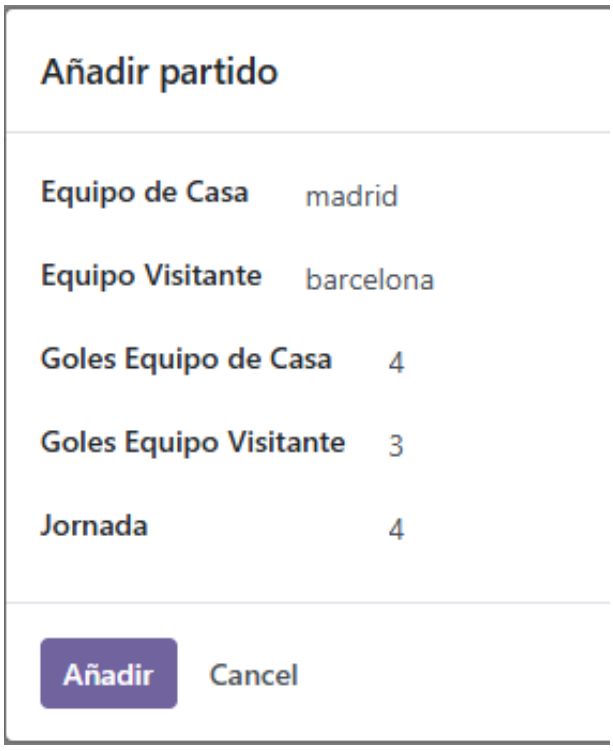
```
acl_liga_partido_wizard,liga.partido_wizard,model_liga_partido_wizard,,1,1,1,1
```

Practica1 > Practica_Docker > addons > EJ07-LigaFutbol > security > ir.model.access.csv

```
1  id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
2  acl_equipo,liga.equipo_default,model_liga_equipo,,1,1,1,1
3  acl_partido,liga.partido_default,model_liga_partido,,1,1,1,1
4  acl_liga_equipo_wizard,liga.equipo_wizard,model_liga_equipo_wizard,,1,1,1,1
5  acl_liga_partido_wizard,liga.partido_wizard,model_liga_partido_wizard,,1,1,1,1
6
```

Con el último ajuste terminado, iremos a nuestro modulo para comprobar su correcta implementación creando un nuevo partido.





Como se puede observar, la implementación fue exitoso.

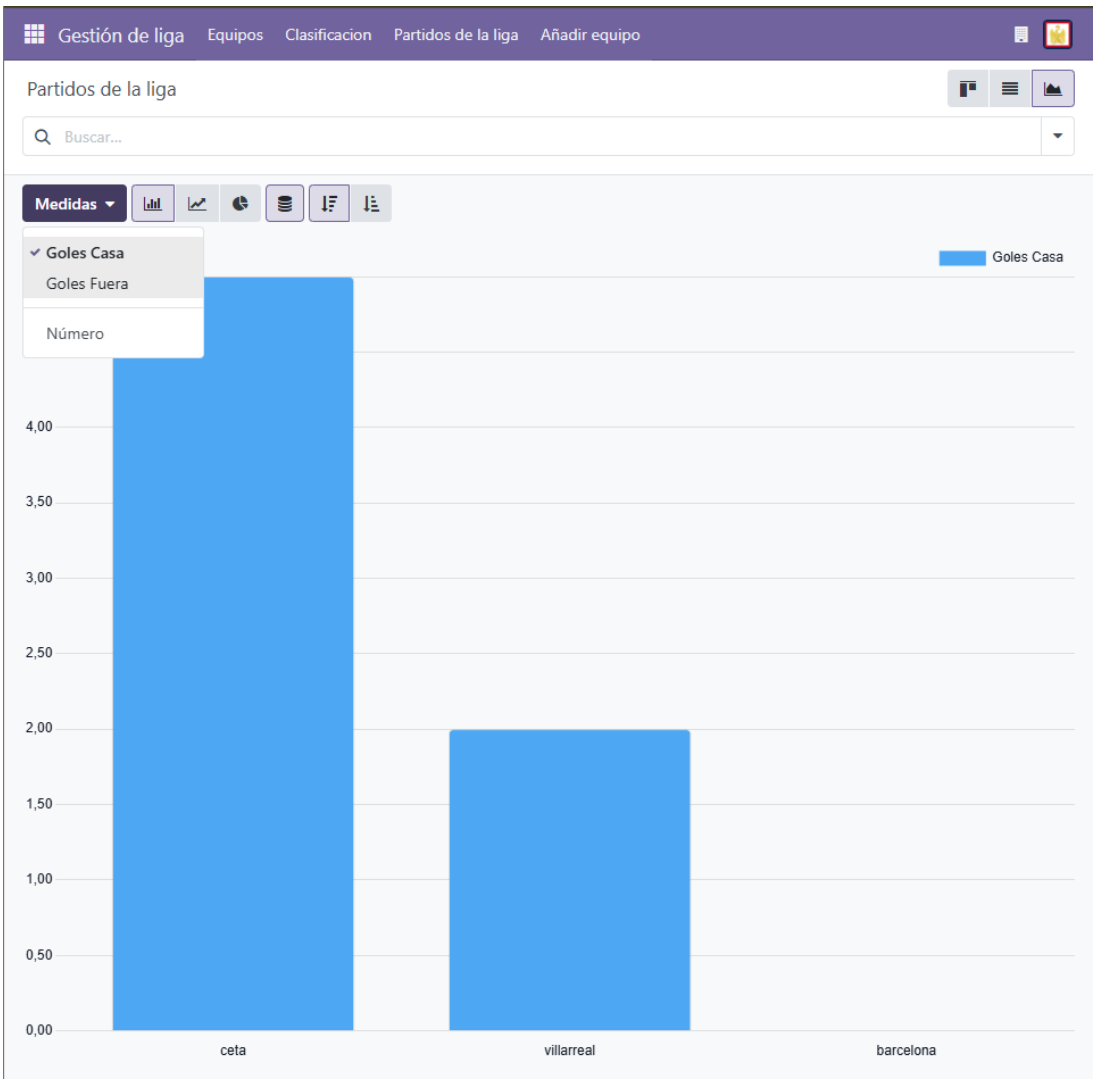
2.6. Vista Graph

Para finalizar con los ejecicios de la **Actividad 1**, en la última parte se nos pide que creemos una nueva vista tipo **Gráfica**. Esto lo realizaremos agregando en el archivo **Liga_partido.xml** este código:

Datos del CSV

```
<!-- Vista Graph -->
<record id="liga_partido_view_graph" model="ir.ui.view">
    <field name="name">Gráfico goles locales</field>
    <field name="model">liga.partido</field>
    <field name="arch" type="xml">
        <graph string="Goles Equipo Local" type="bar">
            <!-- Agrupamos por equipo de casa -->
            <field name="goles_casa" type="measure"/>
            <field name="equipo_casa" type="row"/>
        </graph>
    </field>
```

Con la vista nueva agregada, consultaremos las vistas de los partidos para comprobar su correcto funcionamiento



Con la comprobación realizada, daremos por finalizada la **Actividad 1**

3. Actividad 02

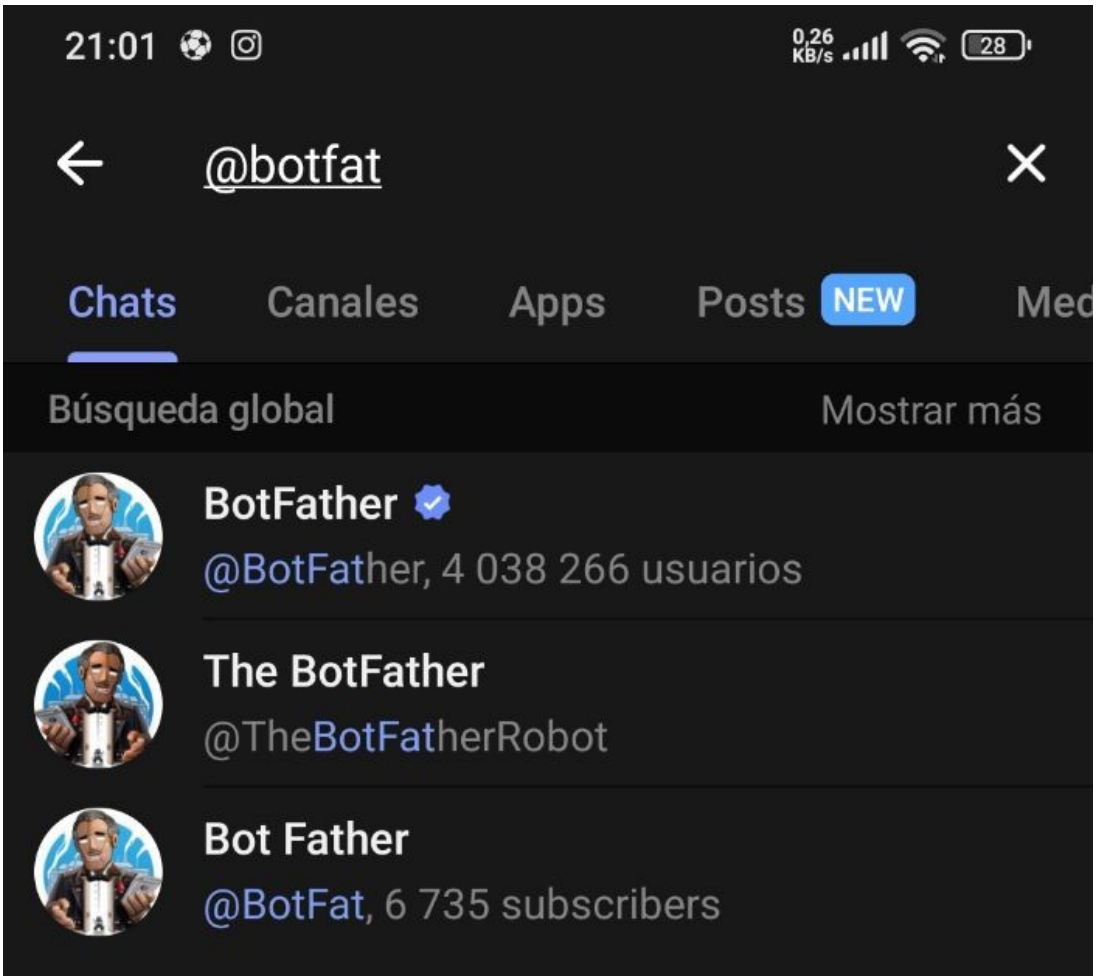
En esta actividad se nos pide probar los diferentes endpoints de la **API** que contiene el modulo **"EJ08-API-REST_Socios"**. Esta prueba se realizara empleando las peticiones **GET, POST, PUT y DELETE**.
Para poder ver el vídeo Ingrese Aquí

4. Actividad 03 – Bot de Telegrama

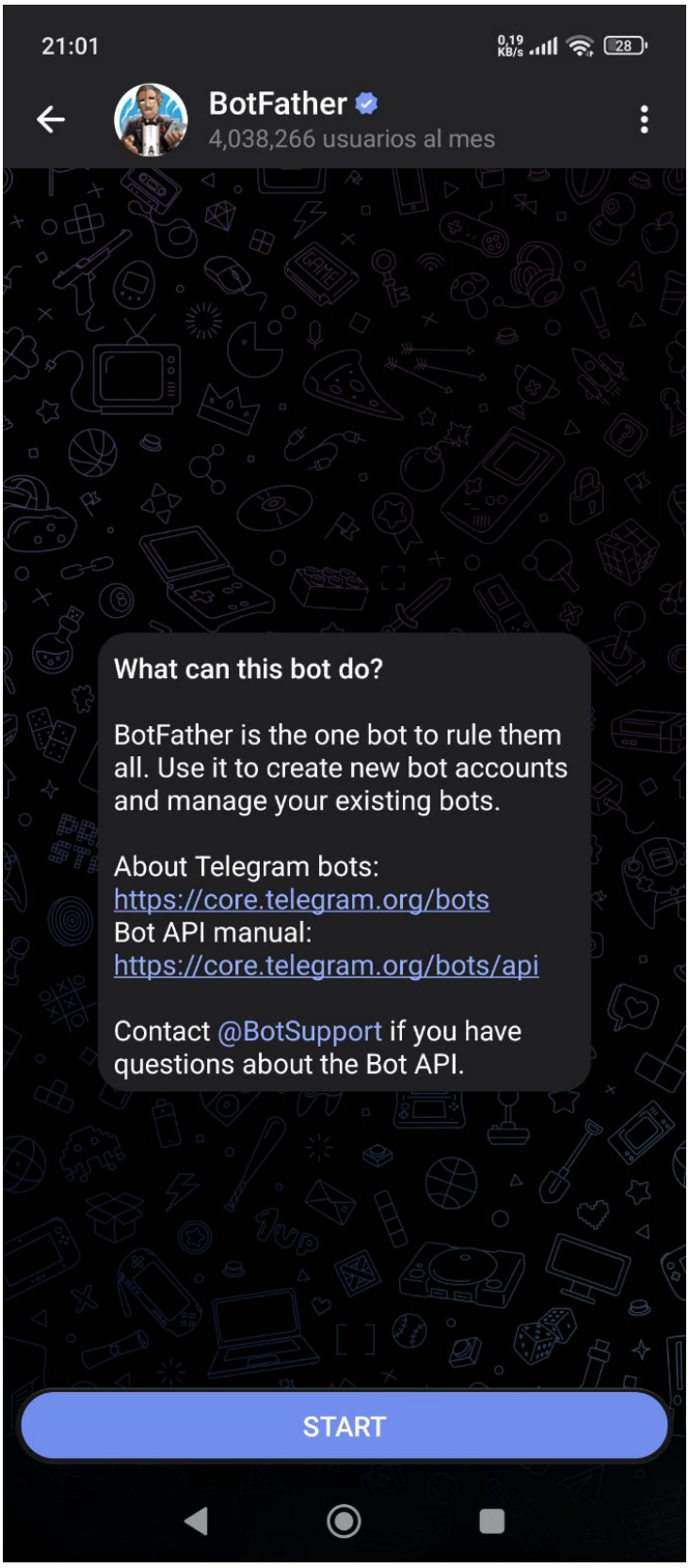
En esta actividad, se nos pide crear un **Bot de Telegrama** empelando las API de la actividad anteriór.

4.1. Configuración de Bot

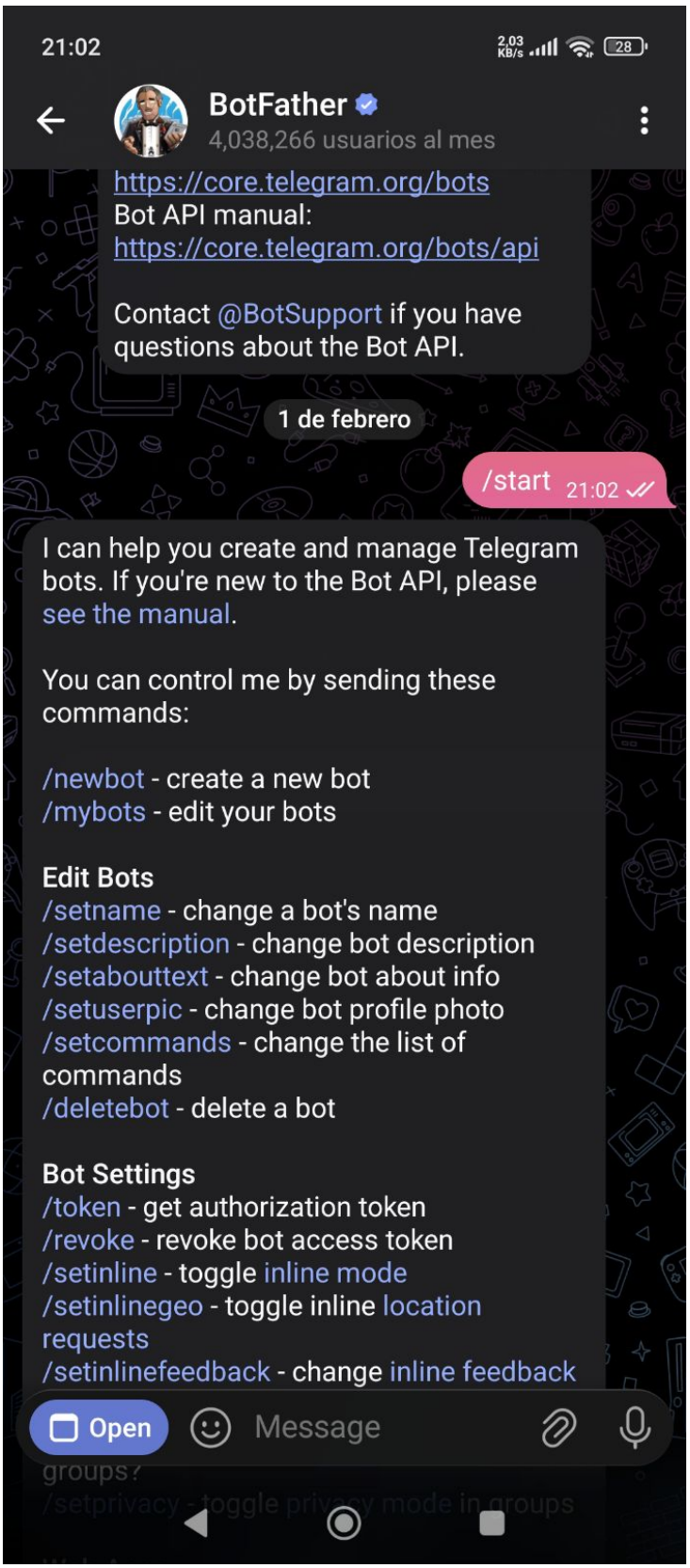
Para comenzar, desde la aplicación de **Telegrama** de nuestro dispositivo, realizaremos una busqueda del bot **@BotFather**



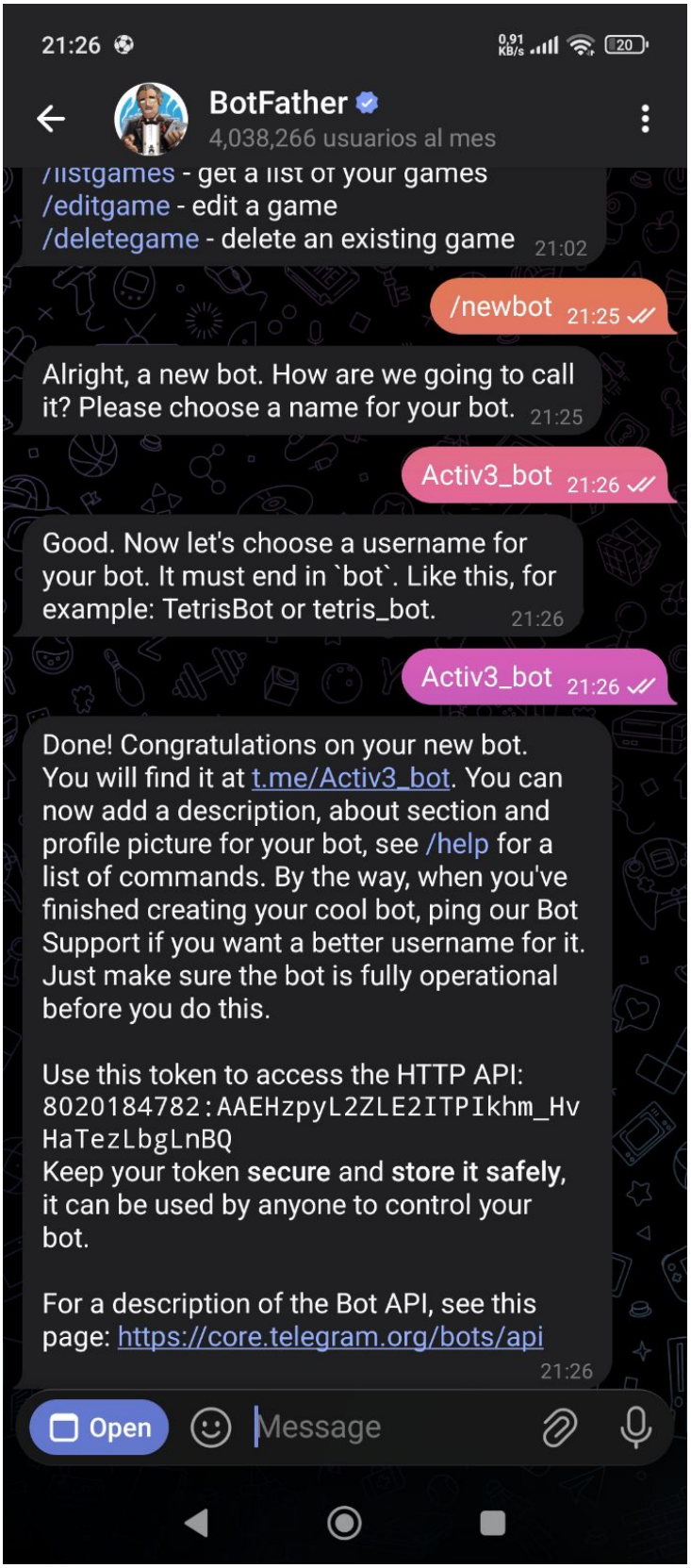
Una vez localizado, entraremos en el chat e iniciamos la conversación.



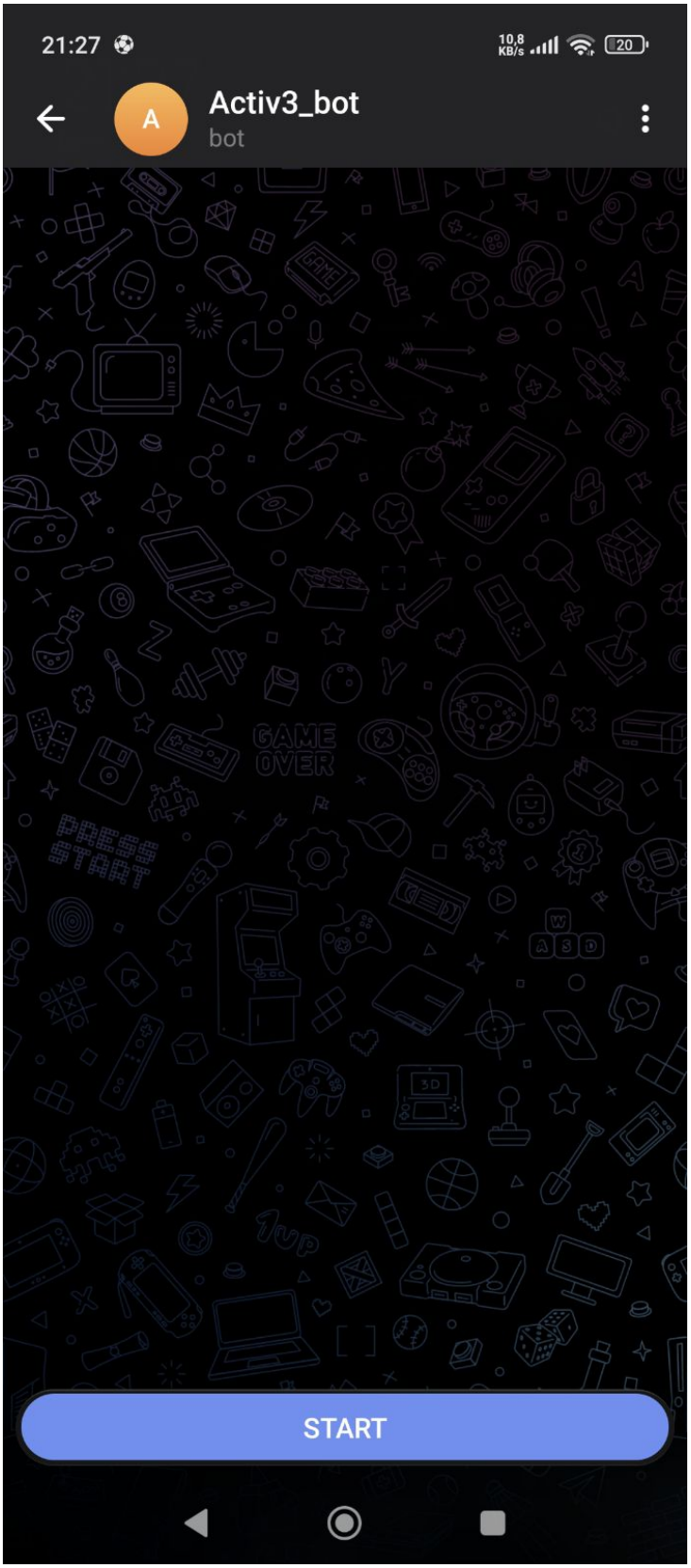
Al iniciar la conversación, no mandará un mensaje de las acciones que podemos realizar. En nuestro caso, nos interesa la sección de nuevo bot (**/NewBot**).



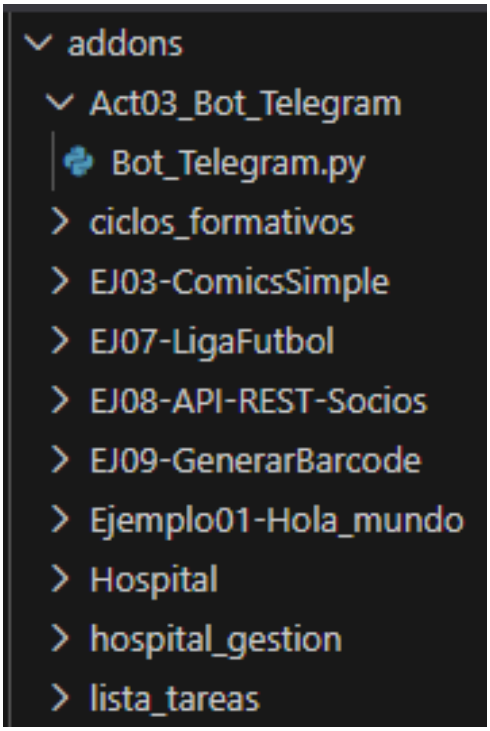
Una vez que seleccionamos **/newbot**, nos pedirá que ingresemos el nombre de este.
Con el nombre seleccionado, el bot nos mandará un mensaje con el enlace al bot que hemos creado asi como un **Token de Acceso desde un API HTTP**



A continuación, entramos al nuevo bot que hemos creado donde nos espera con un botón para iniciarlo.



Con el bot creado, nos dirigimos a la capeta de módulos de nuestro Odoo para crear un nuevo proyecto llamado **Bot_Telegram.py**



Con el archivo creado, antes de comenzar a programar el bot, instalaremos la dependencias necesarias. Para ello, emplearemos los siguientes comando en nuestra terminal.

```
PS C:\Users\bferfer\Desktop\Odoo\Practica1\Practica_Docker> pip install python-telegram-bot requests
Defaulting to user installation because normal site-packages is not writeable
Collecting python-telegram-bot
  Downloading python_telegram_bot-22.6-py3-none-any.whl.metadata (17 kB)
Collecting requests
  Downloading requests-2.32.5-py3-none-any.whl.metadata (4.9 kB)
Collecting httpcore>=1.0.9 (from python-telegram-bot)
  Downloading httpcore-1.0.9-py3-none-any.whl.metadata (21 kB)
Collecting httpx<0.29,>=0.27 (from python-telegram-bot)
  Downloading httpx-0.28.1-py3-none-any.whl.metadata (7.1 kB)
```

Primero con las dependencias de **Telegrama bot y Request**

```
PS C:\Users\bferfer\Desktop\Odoo\Practica1\Practica_Docker> pip install python-dotenv
Defaulting to user installation because normal site-packages is not writeable
Collecting python-dotenv
  Downloading python_dotenv-1.2.1-py3-none-any.whl.metadata (25 kB)
Downloading python_dotenv-1.2.1-py3-none-any.whl (21 kB)
Installing collected packages: python-dotenv
  WARNING: The script dotenv.exe is installed in 'C:\Users\bferfer\AppData\Roaming\Pyth
  Consider adding this directory to PATH or, if you prefer to suppress this warning, us
Successfully installed python-dotenv-1.2.1

[notice] A new release of pip is available: 25.3 -> 26.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Luego con la instalación de la variable de entorno **Dotenv**

Una vez finalizadas las instalaciones, nos dirigiremos a nuestro archivo **.ENV** donde configuraremos una nuevas variables de entorno necesarias para trabajar.

```
TOKEN_BOT_TELEGRAM=8020184782:AAEHzpyL2ZLE2ITPIkhm_HvHaTezLbgLnBQ
URL_API_REST=http://localhost:9001/gestion/apirest/socio
URL_API_REST_ALL_SOCIOS=http://localhost:9001/gestion/socio
```

Como se puede ver, hemos añadido el **token**, que nos entrego el bot, la **url a la api**, así como la api a todos nuestros socios.

Con todos los pasos previos finalizados, nos dispondremos a programar el funcionamiento de nuestro bot. Para ello, comenzamos importando todas las libreras necesarias, asi coma la declaracion de las variable de entorno

```python
# Importamos las librerías necesarias
from telegram import Update
from telegram.ext import ApplicationBuilder, CommandHandler, MessageHandler, filters, ContextTypes
import requests
import json
from dotenv import load_dotenv
import os

# Cargamos las variables de entorno desde el archivo .env
load_dotenv()
# Obtenemos las variables de entorno necesarias
TELEGRAM_TOKEN = os.getenv("TOKEN_BOT_TELEGRAM")
URL_API_REST = os.getenv("URL_API_REST")
URL_API_REST_ALL_SOCIOS = os.getenv("URL_API_REST_ALL_SOCIOS")
```

A continuación en la función del bot, creamos unas variables y un bucle for para dividir los comandos de los datos.

```python
# Definimos la función para el comando /start
async def comandos_inicio_bot(update: Update, context: ContextTypes.DEFAULT_TYPE):
    # Varable para guardar mensajes recibidos del usuario
    mensaje_usuario = update.message.text

    # Uso de try-except para manejar posibles errores
    try:
        # Variable para separar los datos recibidos
        datos_usuario = mensaje_usuario.split(",")
        # Variable para guardar el comando recibido
        comando = datos_usuario[0].strip().lower()
        # Array para guardar los datos del socio
        datos_socio = {}

        # Recoremos los datos recibidos y los guardamos en el array
        for cada_dato in datos_usuario[1:]:
            # Variable para guardar el comando y los datos del socio
            comando_recibido, cada_dato_socio = cada_dato.split("=")
            # Agregamos los datos al array
            datos_socio[comando_recibido.strip()]= cada_dato_socio.strip()
```

Después, crearemos las acciones que realizara el bot en base a los comandos que este reciba.

```
# Comando para agregar un nuevo socio
if comando=="crear":
    # Parseamos los datos del socio a formato JSON y los mostramos por terminal
    datos_socio_json = json.dumps(datos_socio)
    print(f"Datos del socio a crear: {datos_socio_json}")
    # Hacemos la petición POST a la API REST para crear el socio
    respuesta=requests.post(URL_API_REST, json=datos_socio)
    # Mostramos la respuesta de la API REST por terminal
    print(f"Respuesta de la API REST: {respuesta.text}")
    # Enviamos un mensaje al usuario con la respuesta de la API REST
    await update.message.reply_text(f"Socio creado correctamente.\n{respuesta.text}")

# Comando para obtener todos los socios
elif comando=="consultar":
    # Hacemos la petición GET a la API REST para obtener todos los socios
    respuesta=requests.get(f"{URL_API_REST_ALL_SOCIOS}")
    # Mostramos la respuesta de la API REST por terminal
    print(f"Mostrar todos los socios: {respuesta.text}")
    # Enviamos un mensaje al usuario con la respuesta de la API REST
    await update.message.reply_text(f"Lista de socios:\n{respuesta.text}")
```

```
    # Comando para Eliminar un socio
    elif comando=="borrar":
        # Parseamos los datos del socio a formato JSON y los mostramos por terminal
        datos_socio_json = json.dumps(datos_socio)
        print(f"Datos del socio a eliminar: {datos_socio_json}")
        # Hacemos la petición DELETE a la API REST para elimiar el socio
        respuesta = requests.delete(f"{URL_API_REST}?data={datos_socio_json}")
        # Mostramos la respuesta de la API REST por terminal
        print(f"Respuesta de la API REST:{respuesta.text}")
        # Enviamos un mensaje al usuario con la respuesta de la API REST
        await update.message.reply_text(f"Socio eliminado correctamente.\n{respuesta.text}")

    # Comando para Modificar un sociuo
    elif comando=="modificar":
        # Parseamos los datos del socio a formato JSON y los mostramos por terminal
        datos_socio_json = json.dumps(datos_socio)
        print(f"Datos del socio a modificar: {datos_socio_json}")
        # Hacemos la petición PUT a la API REST para modificar el socio
        respuesta = requests.put(URL_API_REST,  json=datos_socio)
        # Mostramos la respuesta de la API REST por terminal
        print(f"Respuesta de la API REST:{respuesta.text}")
        # Enviamos un mensaje al usuario con la respuesta de la API REST
        await update.message.reply_text(f"El socio ha sido modificado de forma correcta: {respuesta.text}")

except Exception as e:
    print("Error en la ejecución del comando:", e)
    await update.message.reply_text(f"Error en la ejecución del comando: {e}")
```

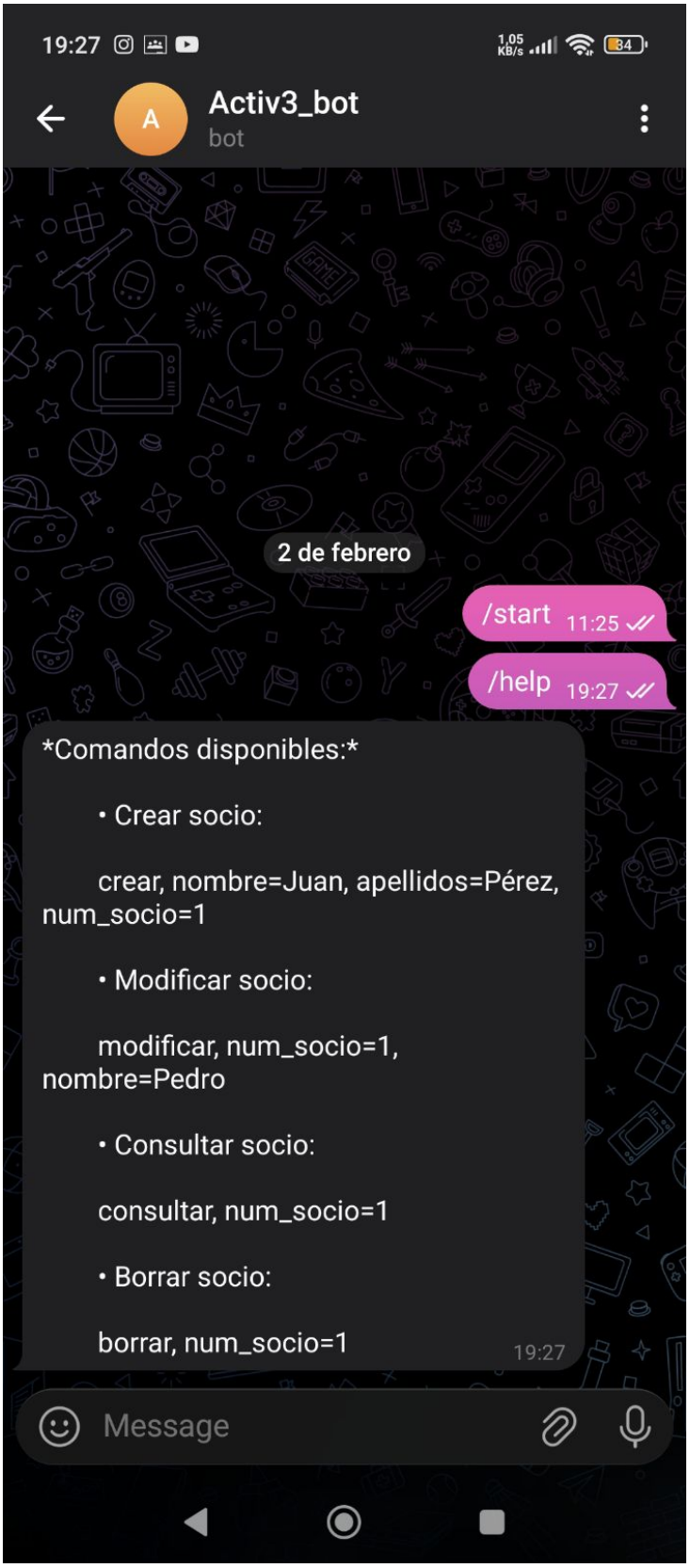
Por último, crearemos dos funciones mas:
Una función **help_bot** para ayudar al usuario y otras función **main** para iniciar el programa.

```python
# Funcio para el comando ayuda para el usuario
async def help_bot(update: Update, context: ContextTypes.DEFAULT_TYPE):
    mensaje_ayuda_user='''
        *Comandos disponibles:*\n
        • Crear socio:\n
        crear, nombre=Juan, apellidos=Pérez, num_socio=1\n
        • Modificar socio:\n
        modificar, num_socio=1, nombre=Pedro\n
        • Consultar socio:\n
        consultar, num_socio=1\n
        • Borrar socio:\n
        borrar, num_socio=1
        '''
    await update.message.reply_text(mensaje_ayuda_user)

# Funcion para iniciar el bot
def main():
    app=ApplicationBuilder().token(TELEGRAM_TOKEN).build()
    # Comando /help
    app.add_handler(CommandHandler("help", help_bot))

    # Comando para el resto de mensaje de texto
    app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, comandos_inicio_bot))

    # Mensaje de Inicio bot por terminal
    print("Bot de Telegram iniciado...")
    app.run_polling()

# Comando de ejecución
if __name__ == "__main__":
    main()
```

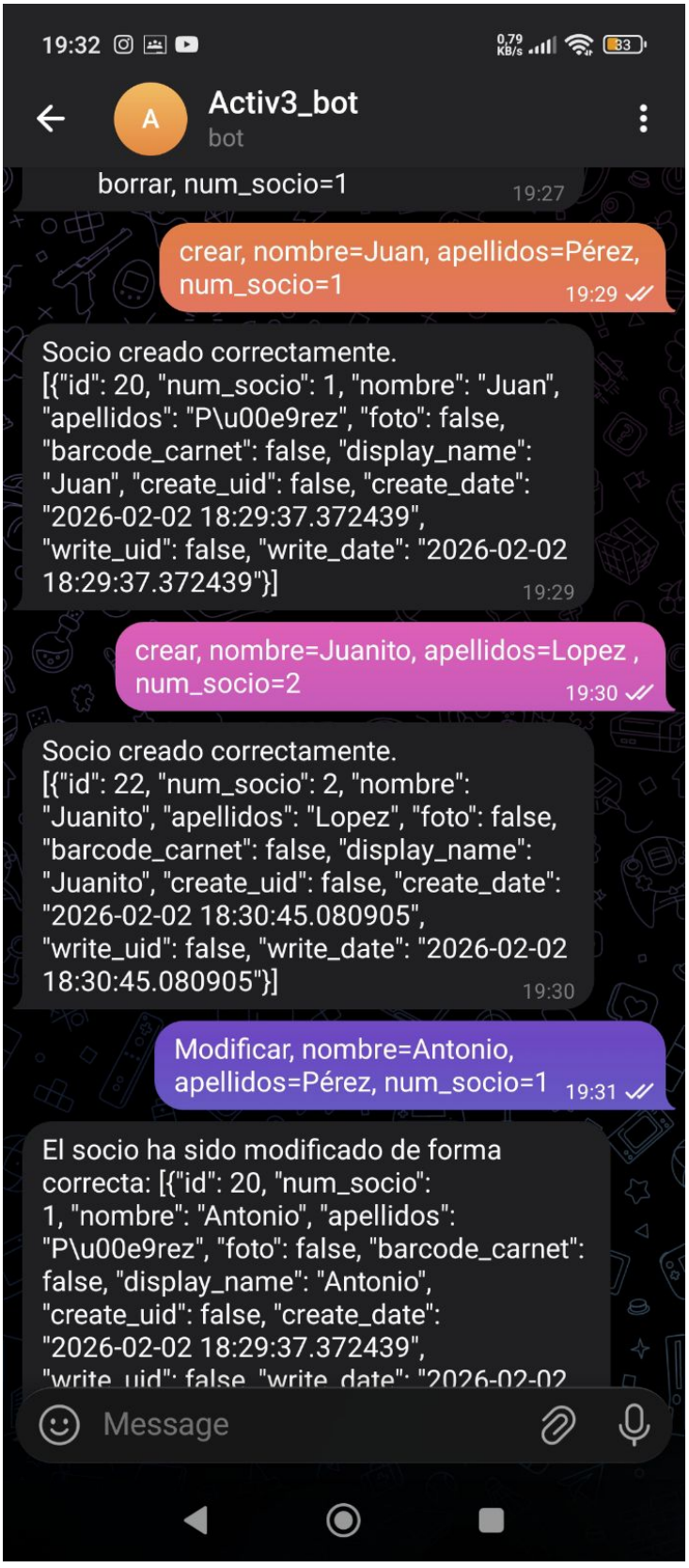
Con todo listo, iniciaremos el programa para interactuar con el bot

```
PS D:\ANA\Desktop\Odoo> python -u "d:\ANA\Desktop\Odoo\Practica1\Practica_Docker\addons\Act03_Bot_Telegram\Bot_Telegram.py"
Bot de Telegram iniciado...
```

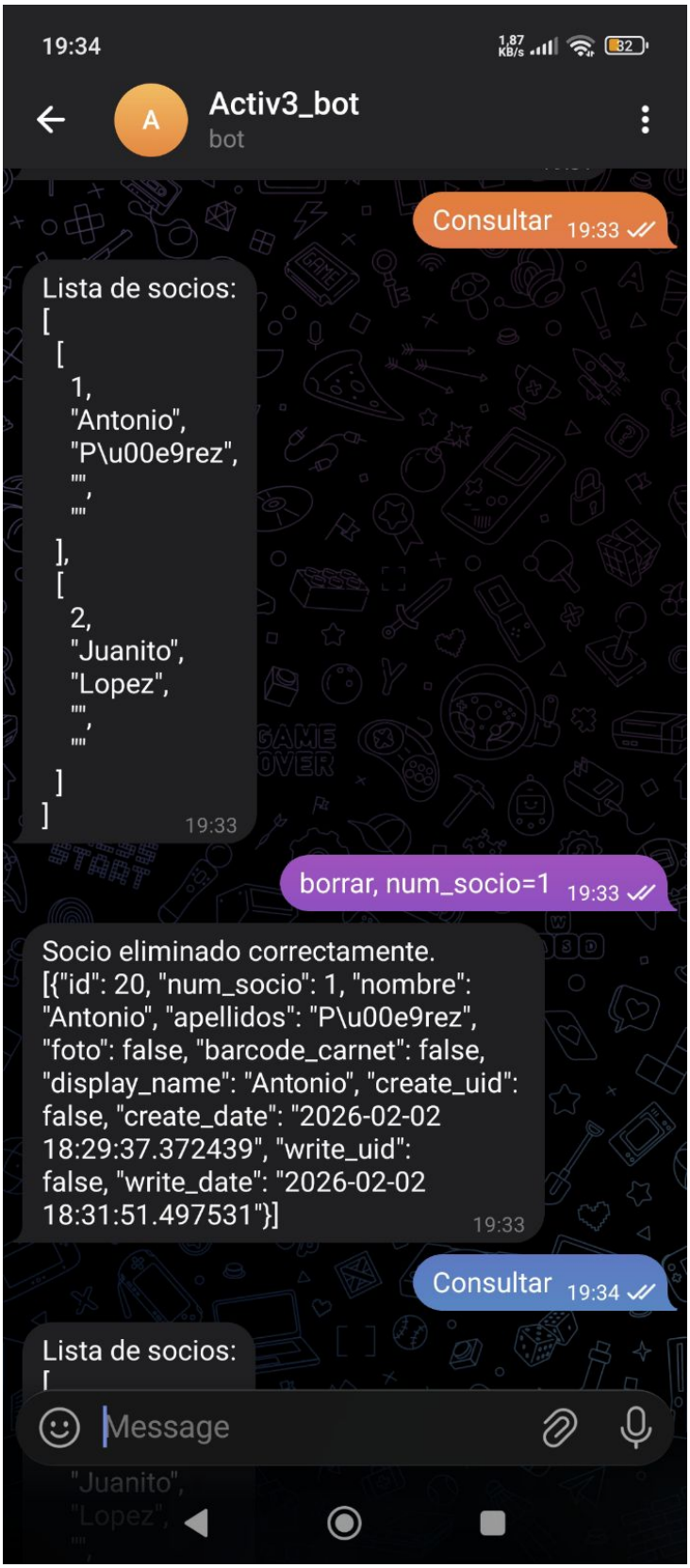
Una vez iniciado el programa, nos dirigiremos al **Chat de Telegram** para comenzar a interactuar con el y comprobar su correcto funcionamiento.
Lo primero que realizaremos en comprobar el correcto funcionamiento de comando **/help**



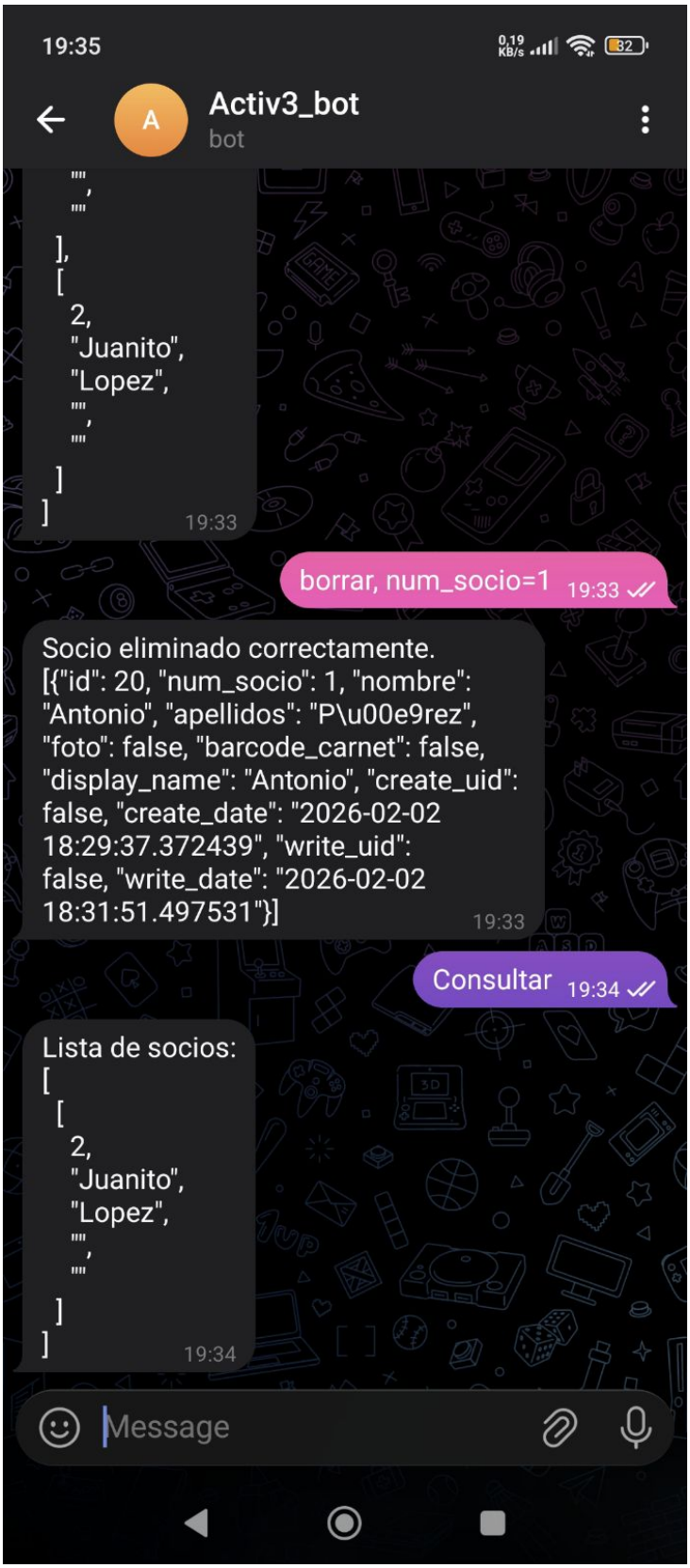
Lo segundo es comprobar la creación de socios asi como la modificación



Con los socios creado y modificados, comprobaremos los socios con una consulta, seguida de una prueba de elimianción de uno de los socios.



Con el mensaje de confirmación, realizaremos una consulta final para comprobar y finalizar con el proyecto.



5. Actividad 04 – Generación de imágenes aleatorias

Para realizar esta actividad en la que deberemos de crear un programa que cree una imagen aleatoria en base a los datos de alto y ancho que le ingresamos desde el url, primero, deberemos de tener instalado el módulo **"EJ09-GenerarBarcode"**.
Con en modulo instalado, tendremos que realizar un paso previo antes de ponernos a trabajar.
Este consiste en instala las librerías que se necesitara para trabajar. Para ello, desde nuestra terminal ingresaremos los siguientes comandos en orden: Lo primero entraremos al contenedor de nuestro Odoo emplenado lo siguinte:

Comando Acceso Contenerdo

```
docker exec -u root -it <id de nustro contenedor>
```

```
PS C:\Users\bferfer\Desktop\Odoo\Practica1\Practica_Docker> docker exec -u root -it 84f450e2b29b bash
root@84f450e2b29b:/# pip list
```

Una vez dentro, ingresaremos el comando para instalar **Barcode** y **Pillow**

Comando de Instalación Barcode

```
python3 -m pip install -break-system-packages python-barcode
```

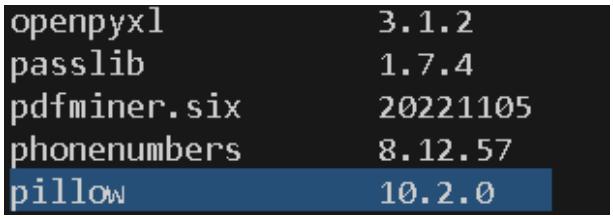
```
root@84f450e2b29b:/# python3 -m pip install --break-system-packages python-barcode
WARNING: Skipping /usr/lib/python3.12/dist-packages/charset_normalizer-3.3.2.dist-info due to invalid metadata entry 'name'
Collecting python-barcode
  Downloading python_barcode-0.16.1-py3-none-any.whl.metadata (2.7 kB)
Downloading python_barcode-0.16.1-py3-none-any.whl (228 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 228.0/228.0 kB 4.0 MB/s eta 0:00:00
WARNING: Skipping /usr/lib/python3.12/dist-packages/charset_normalizer-3.3.2.dist-info due to invalid metadata entry 'name'
Installing collected packages: python-barcode
Successfully installed python-barcode-0.16.1
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager.
```

Con el barcode instalado, utilizaremos el siguinte colando para instla **Pillow** en caso de que no se encuentre:

Comando de Instalación Pillow

```
python3 -m pip install -break-system-packages python-barcode  pillow
```

Tras finaliza, usando el comando **PIP List** comprobaremos la correcta instalación



Con esto, finalizamos con los pasos previos y nos dispondremos a comenzar a codificar el programa.
Para comenzar, nos dirigimos a la carpeta **controllers** de modulo para crear un archivo llamado **imagen_aleatoria.py**
Tras crear el archivo donde se va a trabajar, ingresaremos el siguiente código encargado de generar la imagen de forma aleatoria en base a los parámetros que se le indica

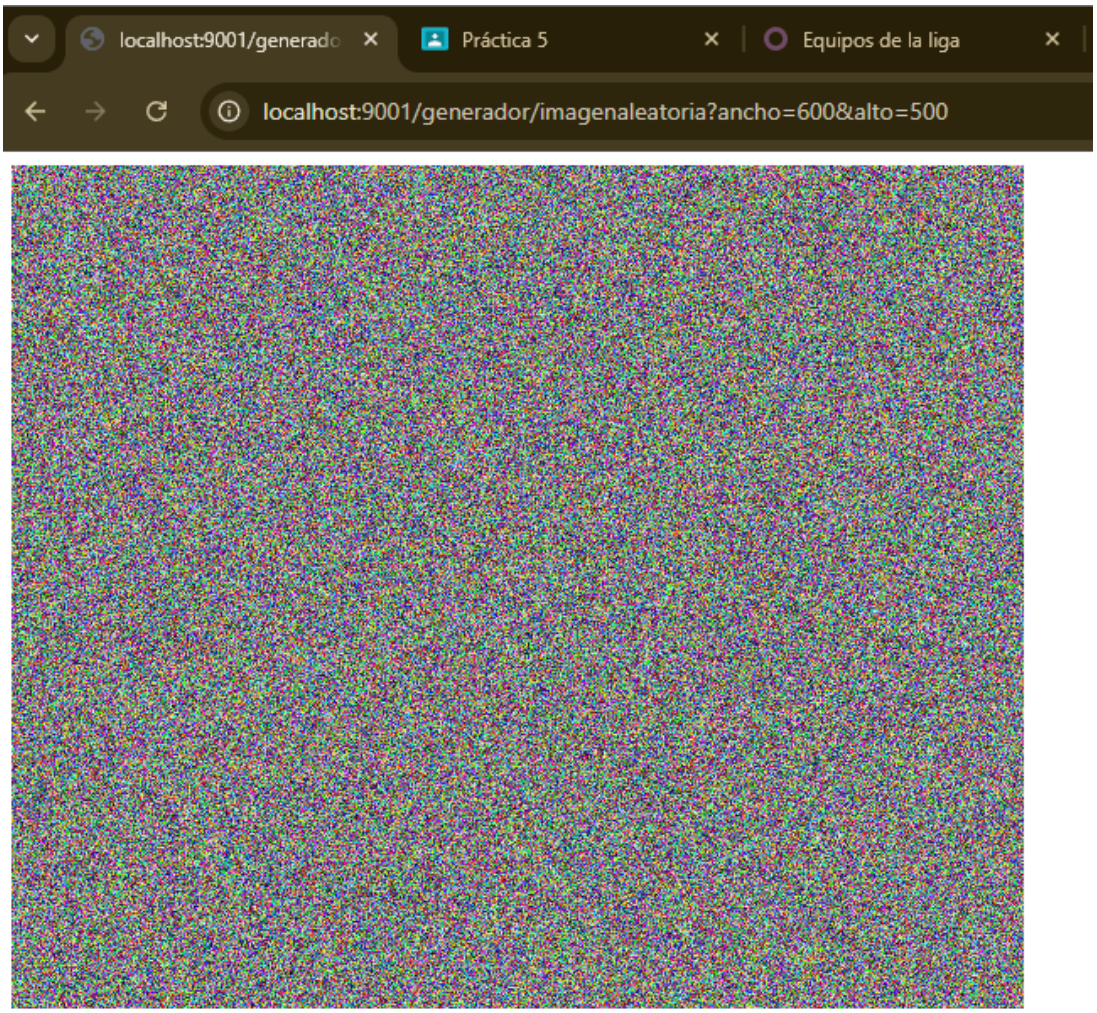
```python
# -*- coding: utf-8 -*-
# Importamos clases necesarias de Odoo para definir controladores HTTP
from odoo import http
from odoo.http import request

# Importamos bibliotecas externas necesarias para generar imágenes en memoria
import base64                    # Para codificar la imagen en base64 (para mostrarla en HTML)
from io import BytesIO           # Para trabajar con flujos de memoria
import random                    # Para generar pixeles de colores aleatorios
from PIL import Image            # Importamos Pillow para generar imágenes

class imagen_aleatoria(http.Controller):
    '''
    Ejemplo de URL: http://localhost:9001/generador/imagenaleatoria?ancho=300&alto=200
    '''
    #Ruta expuesta publicamente  (auth='public'), sin restricciones CORS (cors='*')
    @http.route('/generador/imagenaleatoria',auth='public', cors='*', type='http')
    def crearImagenAleatoria(self, ancho, alto ):
        #covertimos los parametros en enteros
        ancho=int(ancho)
        alto=int(alto)

        #creamos la imagen RGB en la memoria
        imagen=Image.new('RGB',(ancho,alto))
        #Acedemos a los pixeles para colorearlos
        pixeles=imagen.load()

        # Coloreamos cada pixel aleatoriamente
        for pixel_x in range(ancho):
            for pixel_y in range(alto):
                pixeles[pixel_x,pixel_y]=(
                    random.randint(0,255), #Rojo
                    random.randint(0,255), #Verde
                    random.randint(0,255)  #Azul
                )

        # Guardamos la imagen en memoria
        buffer = BytesIO()
        # Determinamos el formto de la imagen en caso de que se desee guardar
        imagen.save(buffer, format='PNG')
        # Empleamos base64.b64encode para convertir los bytes en texto html
        imagen_codificada = base64.b64encode(buffer.getvalue()).decode('utf-8')

        # Devolvemos HTML con la imagen incrustada
        return f'<img src="{imagen_codificada}" />'
```

Por último, antes de realizar una prueba, nos dirigiremos al archivo **___init___.py** donde importaremos nuestro generador.

Practica1 > Practica_Docker > addons > EJ09-GenerarBarcode > controllers >

```
1   # -*- coding: utf-8 -*-
2   from . import generarbarcode
3   #Importamos nuestro generador
4   from . import imagen_aleatoria
```

Tras esto, nos dirigiremos a este Enlace Donde nos generara la siguinte imagen



Con esta ultima actividad damos por finalizado esta documentación