PRACTICA 5

Modulos Odoo Controlador, Herencia y Web Controllers

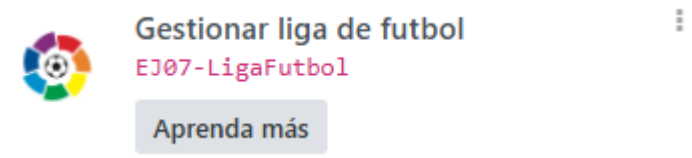
Emilio Corbacho Pereiro

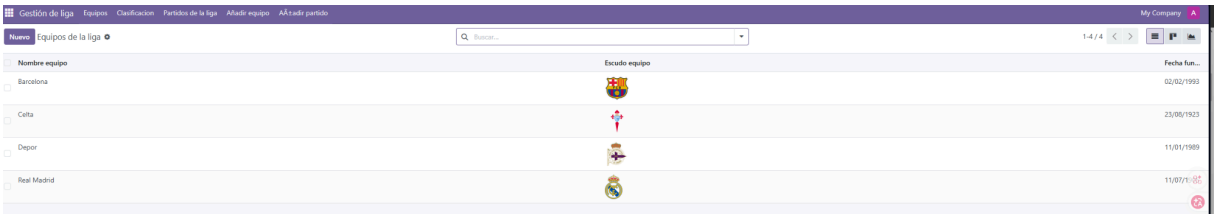
2o Desarrollo de Aplicaciones Multiplataforma

Indice

1. Actividad 01 - Modificacion del modulo EJ07-LigaFutbol

Para comenzar, inserte la carpeta del modulo dentro de addons. Despues, en Odoo con el modo desarrollador activado, actualice la lista de aplicaciones, busque el modulo EJ07-LigaFutbol, lo instale y comprobe que se mostraba correctamente antes de empezar a modificarlo. Tambien hice un commit inicial para poder volver atras con facilidad en caso de errores.





1.1 Reglas de puntuacion especiales

El primer cambio fue localizar donde se gestionaban los puntos de clasificacion. Vi que estaban definidos en liga_equipos.py. Como la nueva logica exigia puntos especiales en caso de goleada, converti el campo de puntos en un campo entero normal para poder actualizarlo durante el procesamiento de cada partido.

```
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)
```

La regla implementada fue la siguiente: si un equipo gana por 4 goles o mas de diferencia, el ganador obtiene 4 puntos y el perdedor recibe -1. En el resto de casos se mantiene el reparto habitual (empate: 1 punto para cada equipo; victoria sin goleada: 3 puntos para el ganador y 0 para el perdedor).

La logica se aplico en la funcion actualizoRegistrosEquipos() del archivo liga_partido.py. Primero reinicie los contadores y variables de puntos, y despues añadi condiciones para distinguir empate, victoria local o visitante y si la diferencia de goles era suficiente para considerar goleada.

```
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.id == recordEquipo.id:

                #Miramos si es victoria o derrota
                if recordPartido.goles_casa>recordPartido.goles_fuera:
                    recordEquipo.victorias=recordEquipo.victorias+1
                    if (recordPartido.goles_casa - recordPartido.goles_fuera) >= 4
                        recordEquipo.puntos = recordEquipo.puntos + 4
```

Por ultimo, los puntos calculados en variables se guardaron en los registros correspondientes de los equipos. Tras actualizar el modulo y probar partidos con diferencias altas de goles, la clasificacion reflejaba correctamente la nueva puntuacion.

```
    #Miramos si es victoria o derrota
    if recordPartido.goles_casa>recordPartido.goles_fuera:
        recordEquipo.victorias=recordEquipo.victorias+1
        if (recordPartido.goles_casa - recordPartido.goles_fuera) >= 4:
            recordEquipo.puntos = recordEquipo.puntos + 4
```

1.2 Botones para alterar los goles de los partidos

En este apartado se pidieron dos botones: uno para sumar 2 goles a todos los equipos locales y otro para sumar 2 goles a todos los equipos visitantes. Para ello cree dos metodos en liga_partido.py, cada uno encargado de recorrer los partidos y actualizar el marcador correspondiente.

```python
def sumar_dos_goles_locales(self):
    for record in self:
        record.goles_casa = record.goles_casa + 2
    self.actualizoRegistrosEquipo()
    return True
```

```python
def sumar_dos_goles_visitantes(self):
    for record in self:
        record.goles_fuera = record.goles_fuera + 2
    self.actualizoRegistrosEquipo()
    return True
```

Despues modifique la vista liga_partido.xml para añadir los botones en la interfaz. Con esto, desde Odoo se podian ejecutar ambas acciones de forma directa.

Durante esta parte tuve problemas para que Odoo detectara los cambios de la vista. Reiniciar el contenedor y actualizar la lista de aplicaciones no fue suficiente, asi que finalmente use el comando de actualizacion del modulo con -u EJ07-LigaFutbol y --stop-after-init, que si aplico los cambios de XML.

```xml
<form>
    <header>
        <button string="+2 goles equipos locales" name="sumar_dos_goles_locales" type="object" class="btn-primary"/>
        <button string="+2 goles equipos visitantes" name="sumar_dos_goles_visitantes" type="object"/>
```

1.3 Web controller para eliminar empates

El objetivo de este punto era crear un controller HTTP accesible desde la ruta http://localhost:8069/eliminarempates para eliminar todos los partidos terminados en empate y devolver el numero de partidos borrados.

Para ello edite el archivo main.py del modulo y añadi una nueva ruta conservando la que ya existia, ya que era necesaria para el funcionamiento previo del ejemplo. El controlador busca los partidos, filtra los que tienen el mismo numero de goles locales y visitantes, cuenta cuantos hay y los elimina.

```python
@http.route('/eliminarempates', type='http', auth='public')
def eliminar_empates(self):
    partidos = request.env['liga.partido'].sudo().search([])
    empates = partidos.filtered(lambda p: p.goles_casa == p.goles_fuera)
    total = len(empates)
    if total:
        empates.unlink()
    return "Partidos eliminados: %s" % total
```

Despues de reiniciar y actualizar el modulo, cree un partido empatado para probarlo. Al acceder a la ruta indicada, el controlador devolvio el resultado esperado y el partido desaparecio del listado.

- Resultado -

Celta : 0
Barcelona : 0

1.4 Informe PDF por cada partido

En este apartado cree un informe PDF basado en QWeb para cada partido. La idea es definir una plantilla XML que recoja los datos del partido y permita imprimirlos en un formato predeterminado.

```
<template id="report_partido_view">
    <t t-call="web.html_container">
        <t t-foreach="docs" t-as="doc">
            <t t-call="web.internal_layout">
                <div class="page">
                    <h2>Partido de liga</h2>
                    <p>
                        <strong>Equipo local:</strong>
                        <span t-field="doc.equipo_casa"/>
                    </p>
                    <p>
                        <strong>Equipo visitante:</strong>
                        <span t-field="doc.equipo_fuera"/>
                    </p>
                    <p>
                        <strong>Resultado:</strong>
                        <span t-field="doc.goles_casa"/> - <span t-field="doc.goles_fuera"/>
                    </p>
                    <p>
                        <strong>Jornada:</strong>
                        <span t-field="doc.jornada"/>
                    </p>
                </div>
            </t>
        </t>
    </t>
</template>
```

Cree un archivo nuevo dentro de la carpeta report (por ejemplo, liga_partido_informe.xml) y defina la accion de informe junto con la plantilla. Despues añadi la ruta del archivo en el manifest, dentro de data, para que Odoo lo cargase al actualizar el modulo.

Una vez actualizado, aparecio la opcion de imprimir el informe del partido desde la interfaz. Al probarla, Odoo genero el PDF correctamente. El formato era funcional, aunque con margen de mejora estetica, que no era el objetivo principal de la practica.

```
    <record id="action_report_partido" model="ir.actions.report">
        <field name="name">Informe partido</field>
        <field name="model">liga.partido</field>
        <field name="report_type">qweb-pdf</field>
        <field name="report_name">EJ07-LigaFutbol.report_partido_view</field>
        <field name="report_file">EJ07-LigaFutbol.report_partido_view</field>
        <field name="binding_model_id" ref="model_liga_partido"/>
        <field name="binding_type">report</field>
    </record>

/odoo>
```

1.5 Wizard para crear nuevos partidos

Para crear partidos mediante un wizard, primero añadi el campo jornada a la clase LigaPartido como fields.Integer. Despues cree los archivos liga_partido_wizard.py y liga_partido_wizard.xml dentro de la carpeta wizard.

```python
class LigaPartidoWizard(models.TransientModel):
    _name = 'liga.partido.wizard'

    equipo_casa = fields.Many2one('liga.equipo', string='Equipo local', required=True)
    equipo_fuera = fields.Many2one('liga.equipo', string='Equipo visitante', required=True)
    goles_casa = fields.Integer()
    goles_fuera = fields.Integer()
    jornada = fields.Integer()

    def add_liga_partido(self):
        ligaPartidoModel = self.env['liga.partido']
        for wiz in self:
            ligaPartidoModel.create({
                'equipo_casa': wiz.equipo_casa.id,
                'equipo_fuera': wiz.equipo_fuera.id,
                'goles_casa': wiz.goles_casa,
                'goles_fuera': wiz.goles_fuera,
                'jornada': wiz.jornada,
            })
```

Tambien añadi la ruta del XML del wizard en el manifest para que Odoo cargase la vista y la accion. La vista del wizard se hizo tomando como referencia el formato de los ejemplos.

```xml
<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='goles_casa'/>
                </group>
                <group>
                    <field name='equipo_fuera'/>
                    <field name='goles_fuera'/>
                    <field name='jornada'/>
                </group>
            </sheet>
            <footer>
                <button string='Crear partido' name='add_liga_partido' class='btn-primary' type='object'/>
                <button string='Cancel' class='btn-default' special='cancel'/>
            </footer>
        </form>
    </field>
</record>
```

Al principio el wizard no funcionaba correctamente. Finalmente detecte que faltaban permisos de acceso, por lo que fue necesario editar ir.model.access.csv y añadir permisos al modelo transitorio del wizard. Tras esa correccion, el wizard ya permitia

crear partidos desde la interfaz.

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

1.6 Vista Graph

Para añadir una vista grafica en partidos, primero modifique la accion principal del modelo para incluir graph en el campo view_mode (por ejemplo: kanban,tree,form,graph).

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

Como el modulo ya tenia una vista grafica en liga_equipo.xml, reutilice ese formato como referencia y cree una vista graph para partidos agrupando por equipo local y sumando los goles locales.

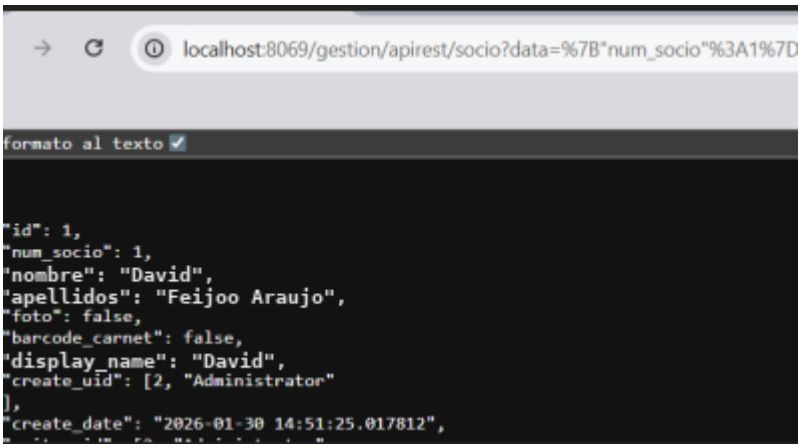
Tras actualizar el modulo, aparecio el icono de grafico en la vista. Al acceder a ella, los datos mostrados coincidian con los partidos registrados.

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

</odoo>
```

2. Actividad 02 - Pruebas del API REST EJ08-API-REST-Socios

En esta actividad active el modulo EJ08-API-REST-Socios y trate de probar sus endpoints desde Postman y otras herramientas similares. Durante varios intentos, los endpoints no devolvian respuesta en Postman, aunque en el navegador si funcionaban correctamente.

Probe distintas opciones para diagnosticar el problema: enviar parametros codificados, pasar la cookie de session_id del navegador, usar Thunder Client en VS Code e incluso Insomnia con varias configuraciones.

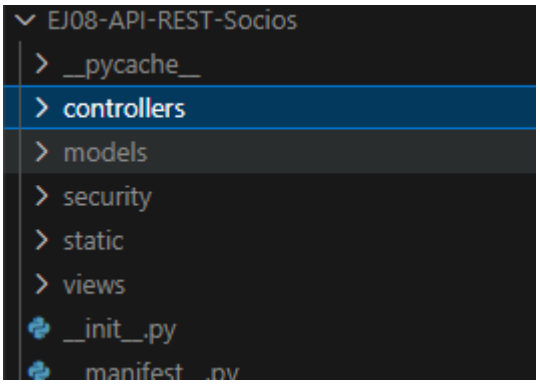


La causa real era que Odoo tenia varias bases de datos disponibles. En el navegador funcionaba porque la sesion ya estaba vinculada a la base de datos correcta por medio de cookies, pero Postman hacia la peticion al puerto sin saber a que base de datos apuntar.

La solucion fue limitar Odoo a la base de datos de trabajo mediante configuracion en odoo.conf, de forma que las peticiones externas apuntasen a la base correcta. Tras aplicar ese cambio, la API empezo a responder correctamente en Postman.

Esta parte me llevo mas tiempo del previsto, pero sirvio para entender mejor como afecta la gestion de multiples bases de datos al probar endpoints HTTP en Odoo.

Finalmente, grabe un video de demostracion del modulo API REST mostrando el funcionamiento y explicando el error encontrado y su solucion.



3. Actividad 03 - Bot de Telegram conectado a la API REST

Para esta actividad empece creando un bot de Telegram con BotFather. Desde Telegram use el comando /newbot, complete la configuracion y guarde el token que genera BotFather.

Despues prepare el entorno en una subcarpeta del proyecto e instale la libreria python-telegram-bot, que es una de las mas utilizadas para este tipo de integraciones. Organice el directorio de forma separada para mantener claro el codigo del bot.

El objetivo del bot era soportar las operaciones Crear, Modificar, Consultar y Borrar consumiendo la API REST del modulo de socios. Para ello cree un archivo principal (bot_socios.py) encargado de gestionar los mensajes del chat, interpretar comandos y lanzar las peticiones a Odoo.

```
TELEGRAM_BOT_TOKEN = os.getenv("TELEGRAM_BOT_TOKEN", "")
ODOO_API_BASE_URL = os.getenv("ODOO_API_BASE_URL", "http://localhost:8070")
```

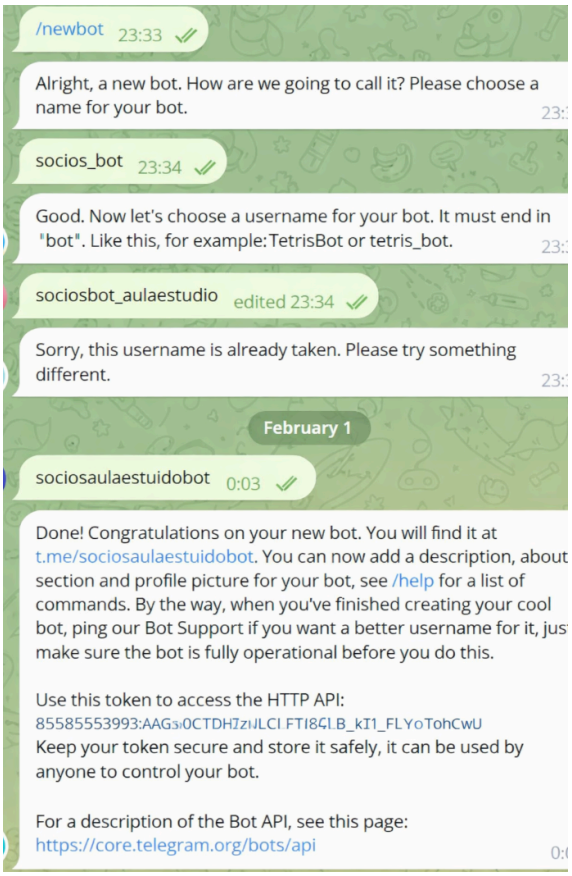
En el codigo defini variables de configuracion como el token del bot, la ruta base de Odoo y la estructura general del endpoint. Tambien implemente una funcion para analizar los mensajes del usuario y extraer la accion y los campos relevantes (nombre, apellidos y num_socio).

```
class ApiClient:
    def __init__(self, base_url: str, timeout: int = 10):
        self.base_url = base_url.rstrip("/")
        self.timeout = timeout
        self.endpoint = f"{self.base_url}/gestion/apirest/socio"

    def _safe_json(self, response: requests.Response) -> Dict:
        try:
            return response.json()
        except ValueError:
            return {}

    def crear_socio(self, nombre: str, apellidos: str, num_socio: str) -> Tuple[boo
        payload = {
            "nombre": nombre,
            "apellidos": apellidos,
            "num_socio": int(num_socio),
        }
```

La funcion de lectura del bot trabaja de forma asincrona, ya que la libreria de Telegram utiliza ese modelo. Cada vez que se recibe un mensaje, se parsea el texto, se comprueba si la orden es soportada, se validan los parametros minimos y se decide que llamada a la API debe ejecutarse.

Ademas, cree funciones separadas para cada operacion de la API (crear, modificar, consultar y borrar), de modo que el codigo fuese mas facil de entender y probar.

Las pruebas mostraron que el bot podia crear y modificar usuarios correctamente devolviendo la informacion esperada, en una salida similar a Postman. En cambio, cuando se enviaban mensajes que no seguian el formato soportado, el bot respondia con el comportamiento previsto.



4. Actividad 04 - Generacion de imagenes aleatorias con Web Controller

Para esta actividad reutilice la estructura del modulo EJ09-GenerarBarcode y añadi un controlador nuevo que, en vez de generar codigos de barras, construye una imagen aleatoria con Pillow (PIL).

Primero comprobe la instalacion de Pillow dentro del contenedor con el comando de pip. En mi caso ya estaba instalado, por lo que no fue necesario añadir mas cambios en ese punto.

Despues cree un archivo nuevo dentro de controllers (por ejemplo, imagen_aleatoria.py) con la logica del endpoint. Este controlador lee los parametros de ancho y alto desde la query, valida que el tamaño este dentro de un rango razonable y genera una imagen RGB con valores aleatorios para cada pixel.

```
class ImagenAleatoriaController(http.Controller):
    @http.route('/imagenaleatoria', auth='public', type='http', methods=['GET'])
    def imagen_aleatoria(self, **kw):
        """
```

Una vez generada la imagen, el controlador la convierte a PNG y devuelve el binario en la respuesta HTTP, de forma que el navegador puede mostrarla directamente.

```
        return http.Response(
            buffer.getvalue(),
            status=200,
            headers=[('Content-Type', 'image/png')],
        )
```

Tambien cree un apartado para las validaciones.

```
    try:
        ancho = int(ancho_raw)
        alto = int(alto_raw)
    except ValueError:
        return http.Response(
            json.dumps({"error": "ancho y alto deben ser enteros"}),
            status=400,
            mimetype='application/json',
        )

    if ancho <= 0 or alto <= 0:
        return http.Response(
            json.dumps({"error": "ancho y alto deben ser mayores que 0"}),
            status=400,
            mimetype='application/json',
        )

    if ancho > MAX_SIZE or alto > MAX_SIZE:
        return http.Response(
            json.dumps({"error": f"Tamano maximo permitido: {MAX_SIZE}x{MAX_SIZ
            status=400,
            mimetype='application/json',
        )
```

Tras reiniciar el servicio, probe la ruta en el navegador con parametros de tamaño y se genero correctamente una imagen aleatoria. Con esto se completaba la ultima actividad de la practica.