IES RIBERA DE CASTILLA

SISTEMAS DE GESTIÓN EMPRESARIAL

CFGS Desarrollo de Aplicaciones Multiplataforma
Informática y Comunicaciones

Módulo ODOO

Curso: *2ºDAM*
Fecha de presentación: *19/01/2026*

Nombre y Apellidos: Daniela Gil Rodríguez
Email: daniela.gilrod@educa.jcyl.es

Índice

1. Introducción

Resumen del Proyecto

El proyecto "**Gestor de Fútbol**" consiste en el desarrollo de un módulo personalizado para el sistema **ERP Odoo**, y esto está diseñado específicamente para la administración integral de entidades deportivas relacionadas con el fútbol.

Este módulo, permite la gestión centralizada de clubes, jugadores, cuerpo técnico (Entrenadores), competiciones y partidos (Local y Visitante). El sistema no solamente actúa como un registro de datos, sino que incorpora lógica de negocio para automatizar varios cálculos (la edad de jugadores o el total de goles que hay en un partido) y valida la integridad de la información (por ejemplo, evita que haya un partido entre el mismo equipo, ya que no es posible esto en la vida real, o que se dupliquen dorsales, ya que un jugador tiene un dorsal fijo en fútbol profesional).

Este módulo, se integra de forma nativa en la interfaz de Odoo, aprovechando el **framework** que tiene potente para ofrecer vistas de lista, formularios detallados y capacidades de búsqueda y de filtrado avanzadas. Además, se expone una API REST muy básica que permite la interacción con sistemas externos, facilitando la consulta y manipulación de los datos de los jugadores desde otras aplicaciones.

Motivación

La gestión de un club de fútbol o de una liga, implica el manejo de una gran cantidad de datos interrelacionados. Los métodos tradicionales (hojas de cálculo .xlsx o documentos en papel) suelen derivar en inconsistencias de datos, pérdida de información y duplicidad de esfuerzos.

La ***motivación principal*** de este proyecto es lo siguiente:

1. **Centralización**: Unificar toda la información deportiva en una única base de datos relacional accesible y segura.
2. **Integridad**: Asegurar que los datos sean coherentes mediante restricciones automáticas (por ejemplo, no puede haber un salario negativo, son positivos).
3. **Automatización**: Reducir la carga administrativa calculando los datos derivados automáticamente.
4. **Escalabilidad**: Crear una base sólida sobre la plataforma Odoo que permita futuras expansiones, como la gestión de ventas de entradas, gestión de socios o de la contabilidad integrada.
5. **Interoperabilidad**: Proveer de una interfaz programática (API) para permitir el desarrollo futuro de aplicaciones móviles o webs para los aficionados sobre su equipo favorito.

Estructura de la Memoria

Se estructura en seis bloques principales:

- **Estado del Arte**: Contextualización de las tecnologías ERP y Odoo.
- **Descripción General**: Objetivos y herramientas utilizadas.
- **Documentación Técnica**: Análisis funcional, diseño de base de datos, código fuente y pruebas.
- **Manuales**: Guías para el usuario final y para el administrador del sistema.
- **Conclusiones**: Reflexión sobre el desarrollo y posibles mejoras.

2. Estado del arte

Definición de ERP

Un sistema de **Planificación de Recursos Empresariales (ERP, en inglés, Enterprise Resource Planning)** es un tipo de software que las organizaciones utilizan para gestionar las actividades empresariales diarias, como la contabilidad, el aprovisionamiento, la gestión de proyectos, la gestión de riesgos y el cumplimiento. Una suite ERP completa también incluye software de gestión del rendimiento empresarial que ayuda a planificar, presupuestar, predecir e informar sobre los resultados financieros de una organización.

Los sistemas ERP enlazan muchos procesos empresariales y facilitan el flujo de datos entre ellos. Al recopilar los datos transaccionales compartidos de múltiples fuentes, los sistemas ERP eliminan la duplicación de datos y proporcionan integridad de datos. Hoy en día, los sistemas ERP son críticos para gestionar miles de negocios de todos los tamaños y en todas las industrias.

Por qué se utiliza Odoo

Para este proyecto he seleccionado **Odoo** por las siguientes razones:

1. **Código Abierto (Open Source)**: La versión Community de Odoo es libre, lo que permite el estudio y modificación profunda de su código sin costes de licencias, ideal para un entorno académico y de desarrollo, ya que es mi situación.
2. **Modularidad**: Odoo se basa en una arquitectura modular. "Gestor de Fútbol" es un módulo más que se acopla al sistema sin afectar a otros módulos instalados (como Ventas o Inventario), permitiendo una instalación y desinstalación limpia.
3. **Framework de Desarrollo Rápido**: Odoo provee un framework potente que abstrae gran parte de la complejidad del desarrollo web (ORM, Vistas XML, Seguridad ACL). Esto permite centrarse en la lógica de negocio (reglas del

fútbol) en lugar de en la infraestructura básica (conexiones a BD, diseño HTML/CSS básico).

4. **Python y PostgreSQL**: Utiliza tecnologías estándar de la industria, robustas y con una gran comunidad de soporte.

3. Descripción general del proyecto

3.1. Objetivos

El **objetivo principal** es desarrollar el módulo **gestor_futbol** para cubrir las siguientes necesidades:

- **Gestión de Clubes**: Registro de información básica (nombre, ciudad, fundación) y visualización rápida de su plantilla y partidos.
- **Gestión de Jugadores**: Creación de fichas técnicas completas. Incluye lógica de negocio para:
    - Calcular automáticamente la edad.
    - Determinar si es "Estrella" (Salario > 1M€).
- **Gestión de Competiciones**: Creación de ligas o copas.
- **Gestión de Partidos**: Registro de encuentros, validación de equipos distintos (Local != Visitante) y cálculo automático de marcadores totales.
- **API de Integración**: Proveer un mecanismo para crear y leer jugadores desde fuera de Odoo (REST API).

3.2. Entorno de trabajo (tecnologías de desarrollo y herramientas)

Para el desarrollo del proyecto se han utilizado las siguientes herramientas y tecnologías:

- **Lenguaje de Programación**: ***Python 3***. Usado para el backend, modelos, lógica de negocio y controladores.
- **Sistema de Gestión de Base de Datos**: ***PostgreSQL***. Motor relacional gestionado a través del ORM de Odoo.
- **Lenguaje de Marcado**: ***XML***. Usado para definir las vistas (formularios, árboles, kanban) y reglas de seguridad.
- **Formato de Intercambio**: ***JSON*** (para la API REST) y CSV (para listas de control de acceso - ACL).
- **IDE**: ***Visual Studio Code***. Elegido por su ligereza y extensiones para Python y XML.
- **Docker**: Utilizado para contenerizar los servicios de ***Odoo*** y ***PostgreSQL***, facilitando el despliegue sin dependencias complejas en el host.
- **Git**: Sistema de control de versiones para gestionar el código fuente.
- **Postman**: Herramienta para probar y validar los endpoints de la ***API REST***.

4. Documentación técnica: análisis, diseño, implementación, pruebas y despliegue

4.1. Análisis del sistema (funcionalidades básicas de la aplicación)

La aplicación permite operaciones CRUD (Crear, Leer, Actualizar, Borrar) y lógica avanzada:

1. Jugadores

- **Datos**: Nombre, Posición, Dorsal, Salario, Fecha Nacimiento.
- **Lógica**:
  - **Edad**: Calculada automáticamente al introducir la fecha de nacimiento.
  - **Estrella**: Check automático si salario > 1.000.000€.
- **Restricciones**: El salario no puede ser negativ o. No puede haber dorsales duplicados en el mismo club.

2. Partidos

- **Datos**: Fecha, Local, Visitante, Marcadores, Competición.
- **Lógica**: Nombre del partido autogenerado (ej: "Madrid vs Barça"). Cálculo de goles totales.
- **Restricciones**: El equipo local y visitante no pueden ser el mismo.

3. API REST Controlador en controllers/main.py (/futbol/players) que permite gestión

externa:

- **GET**: Listado JSON de jugadores.
- **POST**: Creación de jugadores.
- **PUT/DELETE**: Actualización y borrado.
- **Manejo de Errores**: Retorna códigos HTTP (404, 400, 500) y mensajes JSON descriptivos ante fallos.

4.2. Diseño de la base de datos

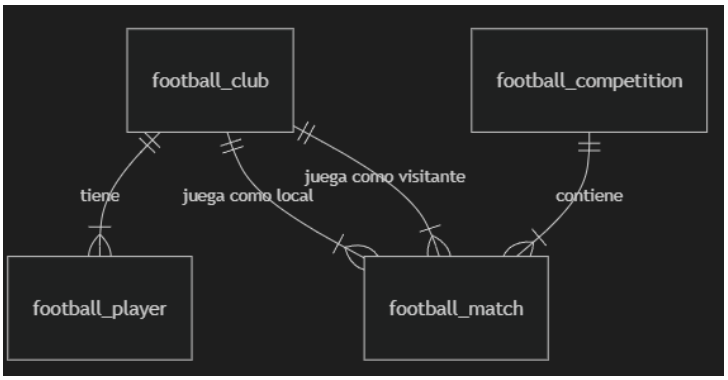
El diseño se basa en un modelo relacional normalizado gestionado por el ORM de Odoo.

Tablas Principales:

- **football_club:**
  - **name** (PK Natural)
  - **foundation_date**, **city**
- **football_player**: Hereda de **res.partner**.
  - **position**, **number**, **salary**, **birthdate**
  - **club_id** (FK -> football_club)
- **football_match**:
  - **home_club_id**, **away_club_id** (FKs -> football_club)

    - **home_score**, **away_score**
    - **competition_id** (FK -> football_competition)
- **football_competition**: name, prize_money.

Diagrama Relacional:



A continuación, vienen las capturas del código, en la siguiente página, para que estén de manera conjunta con sus títulos y fotos.

4.3. Implementación

controllers/main.py:

Esta parte, es la principal del código y del módulo. Ya que se importa todos los modelos y controladores del módulo.

```python
from odoo import http
from odoo.http import request
import json

class FootballController(http.Controller):

    @http.route('/futbol/players', type='http', auth='public', methods=['GET'], csrf=False)
    def get_players(self, **kwargs):
        players = request.env['football.player'].sudo().search([])
        data = []
        for player in players:
            data.append({
                'id': player.id,
                'name': player.name,
                'team': player.club_id.name if player.club_id else None,
                'position': player.position
            })
        return request.make_response(
            json.dumps(data),
            headers=[('Content-Type', 'application/json')]
        )

    @http.route('/futbol/players', type='http', auth='public', methods=['POST'], csrf=False)
    def create_player(self, **post):
        try:
            # Assuming JSON input or Form data
            # For simplicity, reading from post params or json body
            # If body is raw json:
            data = json.loads(request.httprequest.data) if request.httprequest.data else post

            name = data.get('name')
            if not name:
                return request.make_response(
                    json.dumps({'error': 'El nombre es obligatorio'}),
                    status=400,
                    headers=[('Content-Type', 'application/json')]
                )

            player = request.env['football.player'].sudo().create({
                'name': name,
                # Add other fields as needed
            })

            return request.make_response(
                json.dumps({'success': True, 'id': player.id}),
                headers=[('Content-Type', 'application/json')]
            )
        except Exception as e:
            return request.make_response(
                json.dumps({'error': str(e)}),
                status=500,
                headers=[('Content-Type', 'application/json')]
            )

    @http.route('/futbol/players/<int:id>', type='http', auth='public', methods=['PUT'], csrf=False)
    def update_player(self, id, **post):
        try:
            data = json.loads(request.httprequest.data) if request.httprequest.data else post
            player = request.env['football.player'].sudo().browse(id)
            if not player.exists():
                return request.make_response(json.dumps({'error': 'No encontrado'}), status=404)

            player.write(data)
            return request.make_response(json.dumps({'success': True}), headers=[('Content-Type', 'application/json')])
        except Exception as e:
             return request.make_response(json.dumps({'error': str(e)}), status=500)

    @http.route('/futbol/players/<int:id>', type='http', auth='public', methods=['DELETE'], csrf=False)
    def delete_player(self, id):
        try:
            player = request.env['football.player'].sudo().browse(id)
            if not player.exists():
                return request.make_response(json.dumps({'error': 'No encontrado'}), status=404)

            player.unlink()
            return request.make_response(json.dumps({'success': True}), headers=[('Content-Type', 'application/json')])
        except Exception as e:
             return request.make_response(json.dumps({'error': str(e)}), status=500)
```

models/ __init__.py

Importamos todos los archivos creados, que son las clases del módulo.

```
from . import football_club
from . import football_player
from . import football_competition
from . import football_match
from . import football_coach
```

models/football_club.py

Importamos la clase football_club con sus relaciones en el módulo.

```
from odoo import models, fields

class FootballClub(models.Model):
    _name = 'football.club'
    _description = 'Football Club'

    name = fields.Char(string='Nombre del Club', required=True)
    foundation_date = fields.Date(string='Fecha de Fundación')
    city = fields.Char(string='Ciudad')

    player_ids = fields.One2many('football.player', 'club_id', string='Jugadores')
    home_match_ids = fields.One2many('football.match', 'home_club_id', string='Partidos como Local')
    away_match_ids = fields.One2many('football.match', 'away_club_id', string='Partidos como Visitante')
```

models/football_coach.py

Importamos la clase football_coach con sus relaciones en el módulo.

```
from odoo import models, fields

class FootballCoach(models.Model):
    _name = 'football.coach'
    _description = 'Football Coach'

    name = fields.Char(string='Nombre', required=True)
    active = fields.Boolean(default=True)

    club_id = fields.Many2one('football.club', string='Club')

    salary = fields.Float(string='Salario')
    experience_years = fields.Integer(string='Experiencia (Años)')

    licenses = fields.Selection([
        ('c', 'Licencia C'),
        ('b', 'Licencia B'),
        ('a', 'Licencia A'),
        ('pro', 'Licencia Pro')
    ], string='Licencia de Entrenador')
```

models/football_competition.py

Importamos la clase football_competition con sus relaciones en el módulo.

```python
from odoo import models, fields

class FootballCompetition(models.Model):
    _name = 'football.competition'
    _description = 'Football Competition'

    name = fields.Char(string='Nombre de la Competición', required=True)
    prize_money = fields.Float(string='Premio')

    match_ids = fields.One2many('football.match', 'competition_id', string='Partidos')
```

models/football_match.py

Importamos la clase football_match con sus relaciones en el módulo.

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

class FootballMatch(models.Model):
    _name = 'football.match'
    _description = 'Football Match'

    name = fields.Char(string='Nombre del Partido', compute='_compute_name', store=True)
    date = fields.Datetime(string='Fecha', default=lambda self: fields.Datetime.now())

    home_club_id = fields.Many2one('football.club', string='Equipo Local', required=True)
    away_club_id = fields.Many2one('football.club', string='Equipo Visitante', required=True)

    home_score = fields.Integer(string='Goles Local', default=0)
    away_score = fields.Integer(string='Goles Visitante', default=0)

    competition_id = fields.Many2one('football.competition', string='Competición')

    player_ids = fields.Many2many('football.player', string='Jugadores')

    total_goals = fields.Integer(string='Goles Totales', compute='_compute_total_goals', store=True)

    @api.depends('home_club_id', 'away_club_id', 'date')
    def _compute_name(self):
        for record in self:
            home = record.home_club_id.name or 'Unknown'
            away = record.away_club_id.name or 'Unknown'
            date_str = record.date.strftime('%Y-%m-%d') if record.date else ''
            record.name = f"{home} vs {away} ({date_str})"

    @api.depends('home_score', 'away_score')
    def _compute_total_goals(self):
        for record in self:
            record.total_goals = record.home_score + record.away_score

    @api.constrains('home_club_id', 'away_club_id')
    def _check_teams(self):
        for record in self:
            if record.home_club_id == record.away_club_id:
                raise exceptions.ValidationError("El equipo local y el visitante no pueden ser el mismo.")
```

models/football_player.py

Importamos la clase football_player con sus relaciones en el módulo.

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

class FootballPlayer(models.Model):
    _name = 'football.player'
    _inherits = {'res.partner': 'partner_id'}
    _description = 'Football Player'

    partner_id = fields.Many2one('res.partner', required=True, ondelete='cascade')

    position = fields.Selection([
        ('zk', 'Portero'),
        ('df', 'Defensa'),
        ('mf', 'Centrocampista'),
        ('fw', 'Delantero')
    ], string='Posición')

    number = fields.Integer(string='Dorsal')
    salary = fields.Float(string='Salario')
    birthdate = fields.Date(string='Fecha de Nacimiento')

    club_id = fields.Many2one('football.club', string='Club')
    match_ids = fields.Many2many('football.match', string='Partidos')

    age = fields.Integer(string='Edad', compute='_compute_age', store=True)
    is_star = fields.Boolean(string='Es Estrella', compute='_compute_is_star', store=True)

    _sql_constraints = [
        ('unique_number_per_club', 'unique(club_id, number)', '¡El dorsal debe ser único por club!')
    ]

    @api.depends('birthdate')
    def _compute_age(self):
        for record in self:
            if record.birthdate:
                today = date.today()
                record.age = today.year - record.birthdate.year - ((today.month, today.day) < (record.birthdate.month, record.birthdate.day))
            else:
                record.age = 0

    @api.depends('salary')
    def _compute_is_star(self):
        for record in self:
            record.is_star = record.salary > 1000000

    @api.constrains('salary')
    def _check_salary(self):
        for record in self:
            if record.salary < 0:
                raise models.ValidationError("El salario no puede ser negativo.")
```

security/ ir.model.access.csv

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_football_club,football.club,model_football_club,base.group_user,1,1,1,1
access_football_player,football.player,model_football_player,base.group_user,1,1,1,1
access_football_match,football.match,model_football_match,base.group_user,1,1,1,1
access_football_competition,football.competition,model_football_competition,base.group_user,1,1,1,1
access_football_coach,football.coach,model_football_coach,base.group_user,1,1,1,1
```

__manifest__.py

```
{
    'name': 'Gestor de Fútbol',
    'version': '1.0',
    'summary': 'Módulo para gestionar clubes de fútbol, jugadores y competiciones',
    'description': """
        Módulo de Gestión de Fútbol
        ===========================
        Gestiona Clubes, Jugadores, Partidos y Competiciones.
    """,
    'category': 'Deportes',
    'author': 'Student',
    'website': 'https://www.example.com',
    'license': 'LGPL-3',
    'depends': ['base', 'web'],
    'data': [
        # Security
        'security/ir.model.access.csv',
        # Views
        'views/football_menus.xml',
        'views/football_club_views.xml',
        'views/football_player_views.xml',
        'views/football_match_views.xml',
        'views/football_coach_views.xml',
        'views/football_competition_views.xml',
    ],
    'installable': True,
    'application': True,
}
```

4.4. Pruebas de funcionamiento

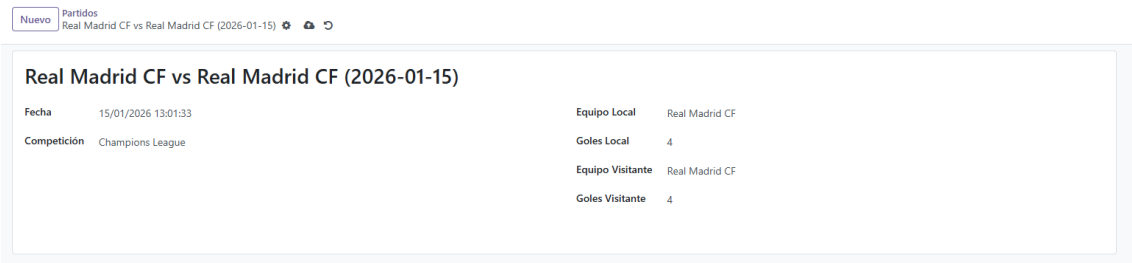
Prueba 1: Validación de API (Postman)

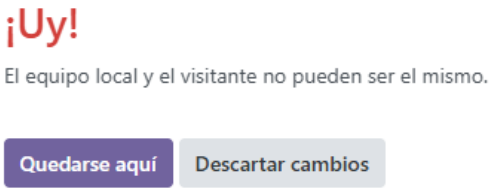
- **Acción:** Petición GET a http://localhost:8069/futbol/players.
- **Resultado Esperado:** Lista JSON de jugadores.
- **Obtenido:**


```
Dar formato al texto ✅
[
  {
    "id": 1,
    "name": "Cristiano Ronaldo",
    "team": "REAL MADRID CF",
    "position": "fw"
  }
]
```


Prueba 2: Validación Lógica (Interfaz)

- **Acción:** Crear un partido "Real Madrid vs Real Madrid".
- **Resultado:** Odoo muestra un popup de error ***ValidationError***: "El equipo local y el visitante no pueden ser el mismo". ***La restricción @api.constrains funciona correctamente***.





4.5. Despliegue de la aplicación

El despliegue de la aplicación **"Gestor de Fútbol"** se ha llevado a cabo en un entorno de servidor local, diseñado para optimizar el ciclo de desarrollo, pruebas y depuración. A continuación, se detalla el procedimiento técnico seguido para la puesta en marcha del sistema.

1. Arquitectura del Despliegue Local

El sistema se ejecuta sobre la máquina del desarrollador (Localhost), donde conviven los tres componentes críticos:

- **Servidor de Base de Datos**: PostgreSQL actuando como backend de persistencia, escuchando en el puerto estándar 5432.
- **Servidor de Aplicación Odoo**: Ejecutando el núcleo de Odoo y nuestro módulo personalizado, escuchando peticiones en el puerto 8069.
- **Cliente Web**: El navegador del usuario accede a la interfaz a través de http://localhost:8069.

2. Configuración del Entorno

Para integrar el módulo gestor_futbol en la instancia de Odoo, se ha modificado la configuración de arranque del servidor.

- **Gestión de Rutas (addons_path)**: Se ha editado el archivo de configuración odoo.conf (o el comando de inicio en Docker/VSCode) para incluir la ruta absoluta del directorio de desarrollo donde reside nuestro módulo.
    - Ejemplo: **addons_path = /usr/lib/python3/dist-packages/odoo/addons,c:\Users\daniela.gilrod\odoo_dev\addons** Esto permite que Odoo reconozca ***gestor_futbol*** como una aplicación instalable al mismo nivel que los módulos nativos.

3. Ejecución de Servicios

El proceso de arranque sigue un orden secuencial estricto:

1. **Inicio de PostgreSQL**: Se levanta el servicio de base de datos para aceptar conexiones.
2. **Inicio de Odoo**: Se ejecuta el script de arranque de Odoo (***odoo-bin***). Durante esta fase, Odoo carga los modelos en memoria y verifica la integridad de los módulos disponibles en el addons_path.
    - En este punto, se monitorizan los logs de la consola en busca de errores de sintaxis Python que pudieran impedir el inicio.

4. Instalación y Activación del Módulo

Una vez los servicios están activos, la instalación final se realiza desde la interfaz web:

1. **Activación del Modo Desarrollador**: Se accede a Ajustes y se activa el modo debug para habilitar herramientas técnicas avanzadas.
2. **Actualización de la Lista de Aplicaciones**: Se fuerza a Odoo a re-escanear el directorio de addons para detectar el nuevo módulo gestor_futbol.
3. **Instalación**: Se localiza el módulo en el menú Aplicaciones y se procede a su instalación. Esto desencadena la creación de las tablas en PostgreSQL (football_player, football_club, etc.) y la carga de las vistas XML.

5. Ciclo de Desarrollo y Actualización

Al tratarse de un entorno de desarrollo, el despliegue es iterativo:

- **Cambios en Código Python**: Requieren un reinicio del servicio Odoo para que el intérprete recargue las clases.
- **Cambios en Vistas XML**: Requieren una actualización del módulo desde la interfaz (botón "Actualizar") para volcar los cambios en la base de datos de vistas.

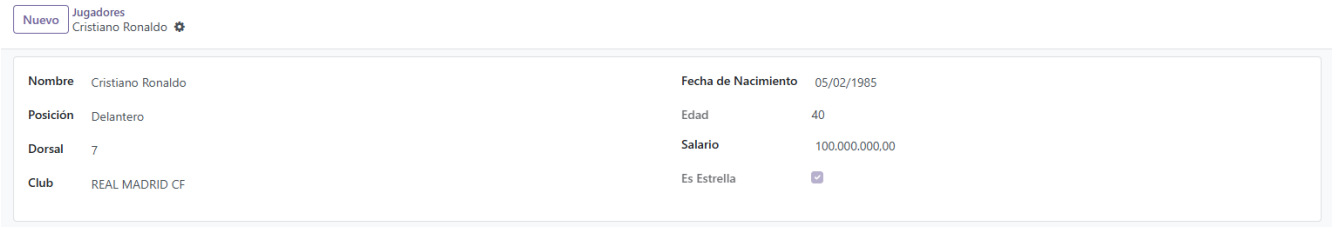
Este enfoque de despliegue local asegura un control total sobre el entorno, permitiendo una depuración inmediata y garantizando la estabilidad antes de un hipotético paso a producción.

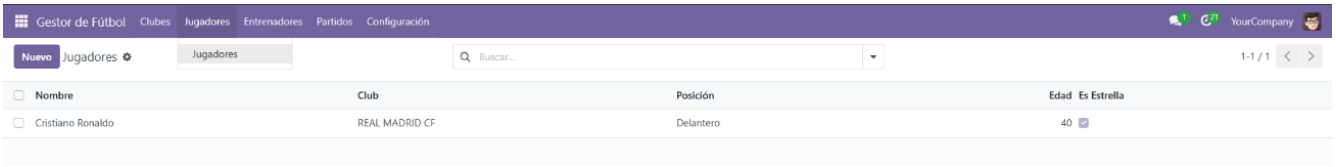
5. Manuales

5.1. Manual de usuario:

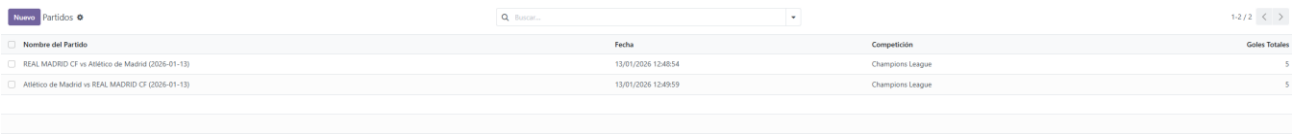
1. **Acceso**: Busca el icono "**Gestor de Fútbol**" en el tablero principal.

2. **Gestión de Jugadores**:
   - Navega al menú "Jugadores".
   - Pulsa "Crear". Rellena Nombre, Club y Salario.
   - Nota: El campo "Edad" se rellenará solo al introducir la fecha de nacimiento.





3. **Partidos**: Utiliza la vista Calendario para ver los encuentros del mes.



5.2. Manual de instalación:

Este manual detalla los pasos necesarios para instalar el módulo "**Gestor de Fútbol**" en una instancia de Odoo ya existente. El procedimiento es válido tanto para entornos locales (**Windows/Linux/Mac**) como para **servidores remotos** (**VPS**, **Cloud**).

Paso 1: Requisitos Previos (Prerequisites)

Antes de comenzar, asegúrese de disponer de lo siguiente:

- **Servidor Odoo**: Versión 14.0 o superior instalada y en ejecución.
- **Acceso al Sistema de Archivos**: Permisos para copiar carpetas en el servidor donde reside Odoo vía FTP, SSH o explorador de archivos.
- **PostgreSQL**: Base de datos activa y vinculada a la instancia de Odoo.

Paso 2: Obtención del Código Fuente

1. **Descarga** el código del módulo, que vendrá en una carpeta llamada gestor_futbol.
   - Nota: Si el archivo es un .zip, descomprímalo. La carpeta resultante debe contener directamente archivos como manifest.py, no otra subcarpeta.
2. **Verifica** que dentro de gestor_futbol existen las carpetas models, views, controllers, etc.

Paso 3: Despliegue en el Servidor

Debes colocar la carpeta del módulo en una ruta que Odoo pueda leer.

1. **Identificar** la carpeta de addons de su instalación.
    - En **Docker**: Usualmente mapeada en **/mnt/extra-addons**.
    - En **Instalación Local**: Generalmente en **.../odoo/server/addons** o una carpeta personalizada definida en su configuración.
2. **Copia** la carpeta **gestor_futbol** completa dentro de ese directorio de addons.
    - ***Ruta final ejemplo: /opt/odoo/custom_addons/gestor_futbol.***

Paso 4: Reinicio del Servicio

Odoo necesita un reinicio para detectar cambios en el **código Python** y archivos nuevos en el sistema.

- **Linux/VPS**: *sudo service odoo restart*
- **Docker**: *docker restart odoo_container*
- **Windows**: *Reinicie el servicio "Odoo Server" desde "Servicios de Windows" (services.msc)*.

Paso 5: Instalación en la Interfaz Web

1. **Accede a Odoo** como usuario **Administrador**.
2. Activa el **Modo Desarrollador**:
    - Vete a **Ajustes** > Desplázate al final > Pulsa "Activar modo desarrollador".
3. **Actualiza la lista de aplicaciones**:
    - Ve al menú **Aplicaciones**.
    - Pulsa la opción "**Actualizar lista de aplicaciones**" (**Update Apps List**) en la barra superior.
    - **Confirma** la **operación** en la ventana emergente.
4. **Instala el módulo**:
    - En la **barra de búsqueda**, **elimina** el filtro "**Aplicaciones**" para ver módulos ajenos.
    - Escribe **gestor_futbol**.
    - Debería de **aparecer** la **tarjeta** del **módulo**. Pulse el botón **Instalar** (o "**Activar**").

Paso 6: Verificación

Una vez finalizada la instalación:

- **Refresca la página del navegador.**
- **Verifica que aparece un nuevo icono llamado "Gestor de Fútbol" en el menú principal.**
- **Entra y prueba a crear un nuevo club para confirmar que la base de datos se ha actualizado correctamente.**

6. Conclusiones y posibles ampliaciones

6.1. Dificultades encontradas durante el desarrollo

El **desarrollo de un módulo** para un sistema **ERP** tan robusto y establecido como **Odoo** presenta una curva de aprendizaje inicial significativa. Durante la ejecución de este proyecto, se ha enfrentado y superado diversos desafíos:

1. **Paradigma de Desarrollo**: Odoo se basa fuertemente en convenciones de nombres y estructuras de directorios. Al inicio, comprender cómo el nombre de un archivo XML o la ubicación de una carpeta models impacta en la carga del módulo. Un simple error tipográfico en el archivo manifest.py o en la lista de dependencias podía causar que el módulo no se cargase, sin errores explicativos claros.
2. **Sintáxis y Herencia de Vistas XML** (QWeb): A diferencia de frameworks web tradicionales donde HTML y la lógica están separados de manera obvia, Odoo utiliza XML para definir la estructura de la interfaz. Entender el mecanismo de herencia de vistas ***(<field name="inherit_id"/>)*** para modificar vistas existentes sin sobrescribirlas fue complejo. Específicamente, ubicar el punto exacto de inyección usando expresiones XPath requirió varias iteraciones de prueba y error.
3. **Complejidad del ORM vs SQL Tradicional**: Adaptarse al ORM de Odoo requirió un cambio de mentalidad. Entender la diferencia entre los métodos create, write y browse, y cómo Odoo gestiona las transacciones de base de datos de manera transparente, fue un punto clave. La gestión de los campos relacionales Many2one y One2many y su contraparte inversa fue especialmente desafiante al diseñar la relación entre Partidos y Clubes (donde un Club aparece dos veces, como Local y Visitante).
4. **Sistema de Permisos** (ACLs): La configuración de seguridad mediante ***ir.model.access.csv*** es crítica. En las primeras pruebas, el módulo fallaba silenciosamente o no mostraba menús debido a la falta de permisos adecuados para el grupo de usuarios. Comprender la granularidad de los permisos (Create, Read, Write, Unlink) fue necesario para solucionar estos problemas de visibilidad.

6.2. Grado de satisfacción y aprendizaje

A pesar de las dificultades iniciales, el grado de satisfacción con el resultado final es alto. El proyecto ha cumplido con todos los objetivos funcionales establecidos por mí y ha demostrado la potencia de utilizar un framework de alto nivel.

Principales aprendizajes adquiridos:

- **Arquitectura Modular**: He aprendido a valorar la importancia de crear software desacoplado. El módulo gestor_futbol puede instalarse en cualquier instancia de Odoo sin romper la funcionalidad existente, lo cual es una lección valiosa de ingeniería de software.

- **Automatización de Procesos de Negocio**: La implementación de campos calculados (como la Edad o el estatus de "Estrella") me ha enseñado cómo el software puede eliminar tareas administrativas repetitivas, aportando valor real al usuario final.
- **Integración de APIs**: La creación de los Controladores REST me ha permitido entender cómo abrir un sistema monolítico al mundo exterior, permitiendo la interoperabilidad con otras tecnologías (como aplicaciones móviles o frontends modernos).

Este trabajo ha servido como una introducción al ecosistema de desarrollo de software empresarial, mostrando que más allá del código, es vital entender las reglas de negocio que se están modelando.

6.3. Posibles ampliaciones futuras

Dado que la base del proyecto es sólida y escalable, existen numerosas vías para continuar su desarrollo y convertirlo en una suite de gestión deportiva profesional de nivel empresarial:

1. **Gestión Avanzada de Eventos del Partido (Acta Digital) Actualmente, el sistema solo registra el resultado final. Una ampliación natural sería crear un modelo football.match.event relacionado con el partido.**
    - ***Funcionalidad***: Permitiría registrar cada incidencia minuto a minuto: goles, tarjetas amarillas/rojas, cambios de jugadores y lesiones.
    - ***Valor***: Esto permitiría generar automáticamente el "Acta del Partido" en PDF e incluso calcular estadísticas avanzadas (posesión, tiros a puerta, rendimiento por jugador).
2. **Módulo de Venta de Entradas (Ticketing) Integración con el módulo nativo de Ventas (Sales) y Sitio Web de Odoo**.
    - ***Funcionalidad***: Definir el aforo del estadio en el modelo del Club y crear "Productos" que sean entradas para los partidos.
    - ***Valor***: Permitiría vender entradas online, gestionar códigos QR para el acceso al estadio y llevar la contabilidad financiera de los ingresos por partido directamente en el módulo de Contabilidad de Odoo.
3. **Portal del Jugador y del Aficionado Utilizando las capacidades de Odoo Website, se podría desarrollar un portal web público.**
    - ***Funcionalidad***: Una página web donde los aficionados puedan ver la tabla de clasificación de la liga actualizada en tiempo real, calendario de partidos y fichas de los jugadores con sus fotos y estadísticas.
    - ***Valor***: Transformaría la herramienta de un sistema puramente administrativo a una plataforma de comunicación con la comunidad de aficionados.

4. **Gestión de Entrenamientos y Fisioterapia Añadir modelos para gestionar el día a día deportivo fuera de los partidos.**
    - ***Funcionalidad***: Planificación de sesiones de entrenamiento, control de asistencia y un módulo de salud para registrar lesiones, tratamientos fisioterapéuticos y tiempos de baja médica.
    - ***Valor***: Ofrecería una visión 360 grados del estado de la plantilla, ayudando al cuerpo técnico a tomar decisiones sobre convocatorias basadas en datos de salud reales.
5. **Gamificación (Liga Fantástica) Crear un sistema de puntuación automático para los usuarios del sistema.**
    - ***Funcionalidad***: Los usuarios podrían crear sus propios equipos virtuales eligiendo jugadores reales de la base de datos. Basado en las estadísticas reales (goles, partidos jugados) registradas en el sistema, los usuarios sumarían puntos.
    - ***Valor***: Añadiría una capa de entretenimiento y compromiso con la plataforma.

Referencias

Documentación Oficial y Guías Técnicas

Odoo S.A. (2024). **Odoo 14 Development Documentation**. Recuperado de: https://www.odoo.com/documentation/14.0/developer/

Referencia principal para la estructura de módulos, ORM y Vistas XML.

PostgreSQL Global Development Group. (2024). PostgreSQL 13 Documentation. Recuperado de: https://www.postgresql.org/docs/13/

Utilizado para consultas sobre tipos de datos y diseño relacional.

Python Software Foundation. (2024). Python 3.8 Documentation. Recuperado de: https://docs.python.org/3.8/

Guía de referencia del lenguaje para la implementación de lógica de negocio.

Recursos de la Comunidad y Tutoriales

Docker Inc. (2024). Docker Hub: Odoo Official Image. https://hub.docker.com/_/odoo

Instrucciones para el despliegue del contenedor odoo:14.0.

Reis, D. (2018). Odoo Development Essentials (4th Edition). Packt Publishing.

Libro de consulta para patrones de diseño avanzados en Odoo.

Herramientas de Desarrollo

Visual Studio Code. Python Extension for VS Code. https://marketplace.visualstudio.com/items?itemName=ms-python.python

Postman API Platform. Learning Center. https://learning.postman.com/docs/getting-started/introduction/

Utilizado para validar las pruebas de la API REST.