



IES RIBERA DE CASTILLA

PROYECTO SGE

CFGS Desarrollo de Aplicaciones Multiplataforma
Informática y Comunicaciones

<Practica11_modulo_Odoo>

Año: <2026>

Fecha de presentación: (1/01/2026)

Nombre y Apellidos: Ángel Maroto García

Email: angel.margar@educa.jcyl.es

1. Introducción

El proyecto de Gimnasio Angel consiste en el desarrollo de un módulo del ERP Odoo, diseñado para la gestión integral de un centro deportivo. El módulo permite la administración de socios, monitores, equipamiento, actividades y sesiones dirigidas por los monitores. Además, incorpora lógica de negocio avanzada como el control automático de aforo y una API REST para consultas externas.

Motivación: La motivación principal es solucionar los posibles problemas que puedan surgir a la hora de gestionar un gimnasio, así como problemas de gestión del aforo o la falta de trazabilidad de los materiales usados.

The screenshot shows the Odoo interface for the 'Gimnasio Angel' project. The top navigation bar includes 'Gimnasio Angel', 'Gestión', and 'Configuración'. Below this, a sub-menu bar has 'Nuevo' highlighted, followed by 'Clases Programadas' and a gear icon. The main content area displays a class entry for 'Zumba' on '2026-01-17 15:00:00'. The entry includes the monitor's name ('Pablo Dominguez') and the number of available spots ('Libres: 6'). At the bottom, a detailed view of the 'Clases Programadas' list is shown, listing the session ID, instructor, start date, and available spots.

Identificador Sesión	Instructor	Fecha Inicio	Libres
Zumba - 2026-01-17 15:00:00	Pablo Dominguez	17/01/2026 16:00:00	6

2. Estado del arte

Definición de ERP: Un ERP es un sistema de software que integra las operaciones principales de una empresa, como recursos humanos, inventario y ventas, en una única base de datos.

Explicar por qué se utiliza Odoo: Se ha elegido Odoo por ser una suite de aplicaciones de código abierto, su arquitectura permite desarrollar módulos que heredan funciones del núcleo, sin modificar el código base y facilitando el mantenimiento y la actualización.

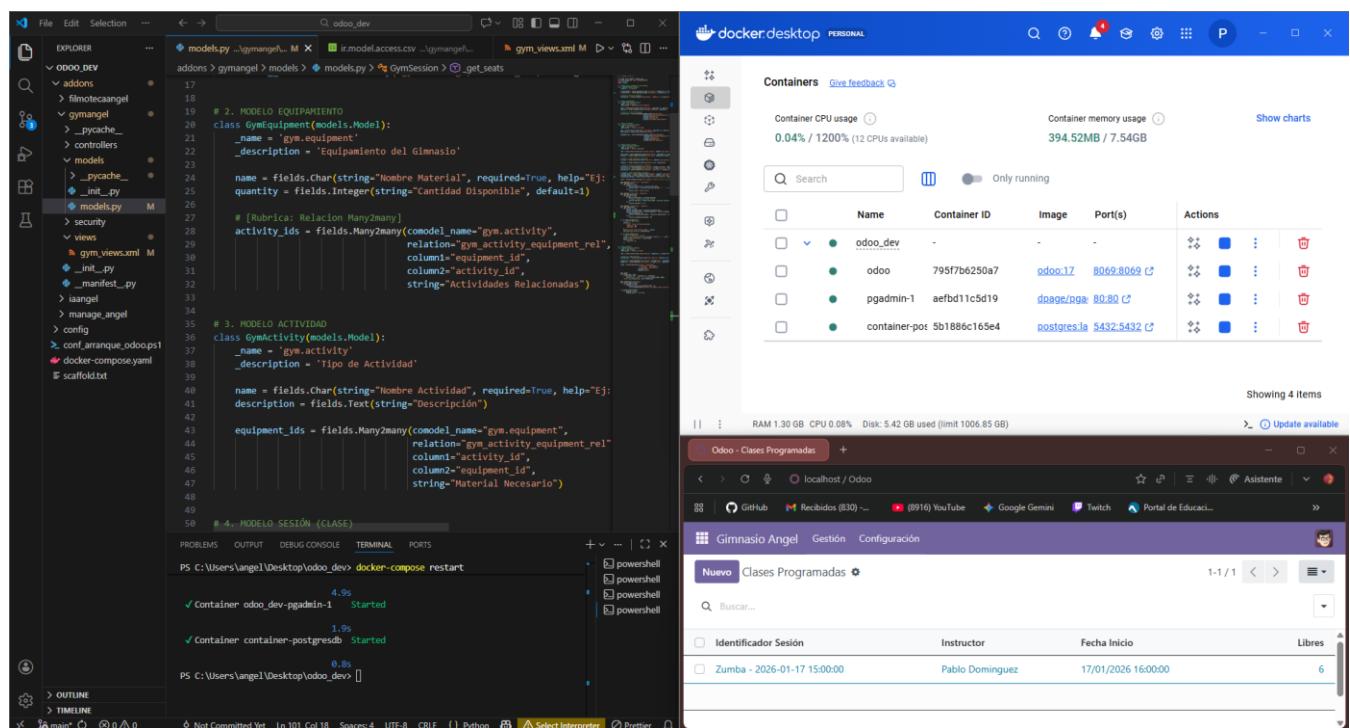
3. Descripción general del proyecto

3.1. Objetivos

- Gestionar una base de datos unificada de socios y monitores.
- Controlar el inventario de equipamiento deportivo.
- Programar sesiones con fecha, hora y capacidad máxima.
- Implementar restricciones para evitar reservas si el aforo está completo
- Desarrollar una API REST para exponer los horarios de la clase.

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

- Virtualización: Docker y Docker Compose para desplegar Odoo y PostgreSQL
- IDE: Visual Studio Code
- Lenguajes: Python (lógica backend) y XML para la definición de las vistas
- SO: Windows con el subsistema de Linux para windows.



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

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

La aplicación permite a un administrador dar de alta actividades y asignarles equipamiento necesario (relación Many2many). Posteriormente, se planifican sesiones (clases) asignando un monitor y una fecha. Los socios pueden inscribirse hasta completar el aforo.

4.2. Diseño de la base de datos

Gym.session: Tabla principal (Nombre, fecha, capacidad)

Gym.booking: Tabla de unión entre Sesión y socio

Gym.activity: Tipos de clase

Res.partner (heredad): Se añaden los campos booleanos is_gym_member e is_instructor

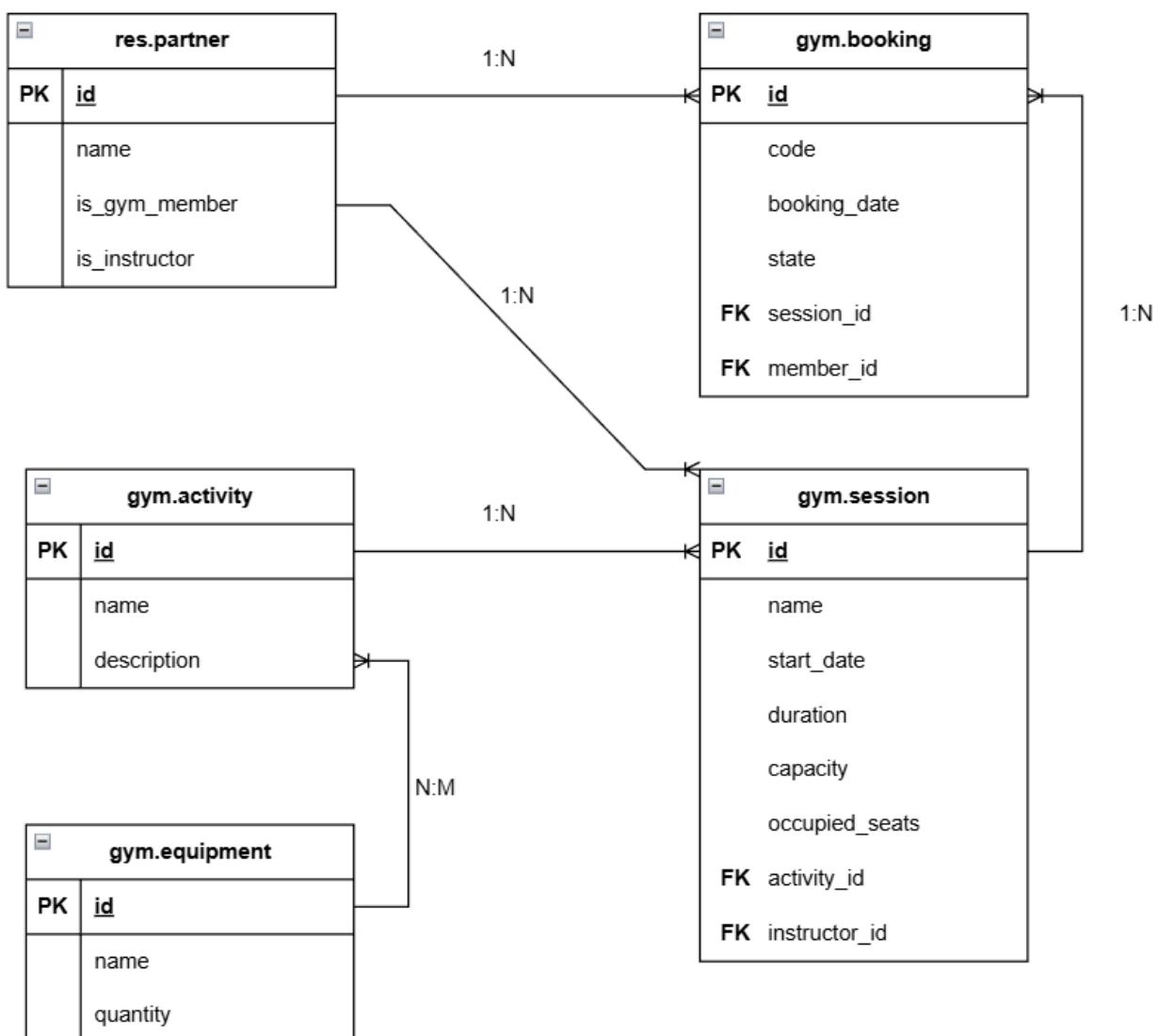
Un socio puede tener muchas reservas.

Una sesión tiene muchas reservas

Una actividad puede tener muchas sesiones

Un monitor puede impartir muchas sesiones

Una actividad usa muchos materiales y un material se usa en muchas actividades



4.3. Implementación

El modulo sigue la estructura estándar de Odoo:

- Models: Contiene las clases de Python y la lógica de negocio

```
addons > gymangel > models > models.py > ...
1  # -*- coding: utf-8 -*-
2
3  from odoo import models, fields, api
4  from odoo.exceptions import ValidationError
5  import datetime
6  import random
7
8  # 1. MODELO SOCIO (HERENCIA DE RES.PARTNER)
9  class GymMember(models.Model):
10     _inherit = 'res.partner'
11
12     is_gym_member = fields.Boolean(string="¿Es Socio?", default=False, help="Marcar si esta persona es socio del gimnasio")
13     is_instructor = fields.Boolean(string="¿Es Monitor?", default=False, help="Marcar si trabaja como monitor")
14
15     # [Rubrica: Relacion One2many]
16     booking_ids = fields.One2many('gym.booking', 'member_id', string="Reservas")
17
18
19 # 2. MODELO EQUIPAMIENTO
20 class GymEquipment(models.Model):
21     _name = 'gym.equipment'
22     _description = 'Equipamiento del Gimnasio'
23
24     name = fields.Char(string="Nombre Material", required=True, help="Ej: Pesas, Esterilla, Bicicleta")
25     quantity = fields.Integer(string="Cantidad Disponible", default=1)
26
27     # [Rubrica: Relacion Many2many]
28     activity_ids = fields.Many2many(comodel_name="gym.activity",
29                                     relation="gym_activity_equipment_rel",
30                                     column1="equipment_id",
31                                     column2="activity_id",
32                                     string="Actividades Relacionadas")
33
34
35 # 3. MODELO ACTIVIDAD
36 class GymActivity(models.Model):
37     _name = 'gym.activity'
38     _description = 'Tipo de Actividad'
39
40     name = fields.Char(string="Nombre Actividad", required=True, help="Ej: Yoga, Zumba, Boxeo")
41     description = fields.Text(string="Descripción")
42
43     equipment_ids = fields.Many2many(comodel_name="gym.equipment",
44                                     relation="gym_activity_equipment_rel",
45                                     column1="activity_id",
46                                     column2="equipment_id",
47                                     string="Material Necesario")
48
```

```

# 4. MODELO SESIÓN (CLASE)
class GymSession(models.Model):
    _name = 'gym.session'
    _description = 'Sesión Programada'

    name = fields.Char(string="Identificador Sesión", compute="_get_name", store=True)

    start_date = fields.Datetime(string="Fecha Inicio", required=True, default=fields.Datetime.now)
    duration = fields.Integer(string="Duración (min)", default=60, help="Duración en minutos")

    # [Rubrica: Campo computado almacenado]
    end_date = fields.Datetime(string="Fecha Fin", compute="_get_end_date", store=True)

    capacity = fields.Integer(string="Capacidad Máxima", default=10)

    # [Rubrica: Relaciones ManyZone y One2many]
    activity_id = fields.Many2one("gym.activity", string="Actividad", required=True)
    instructor_id = fields.Many2one("res.partner", string="Instructor", domain="[(is_instructor, '=', True)])")
    booking_ids = fields.One2many("gym.booking", "session_id", string="Asistentes")

    # [Rubrica: Campos computados avanzados]
    occupied_seats = fields.Integer(string="Ocupadas", compute="_get_seats", store=True)
    available_seats = fields.Integer(string="Libres", compute="_get_seats", store=True)
    occupied_percentage = fields.Float(string="Porcentaje Ocupación", compute="_get_seats", store=True)

    color = fields.Integer(string="Color Kanban") # Necesario para la vista Kanban

@api.depends('activity_id', 'start_date')
def _get_name(self):
    for session in self:
        if session.activity_id and session.start_date:
            session.name = str(session.activity_id.name) + " - " + str(session.start_date)
        else:
            session.name = "Nueva Sesión"

@api.depends('start_date', 'duration')
def _get_end_date(self):
    for session in self:
        if session.start_date and session.duration:
            # Usamos timedelta para sumar minutos
            session.end_date = session.start_date + datetime.timedelta(minutes=session.duration)
        else:
            session.end_date = False

@api.depends('booking_ids', 'capacity')
def _get_seats(self):
    for session in self:
        session.occupied_seats = len(session.booking_ids)
        session.available_seats = session.capacity - session.occupied_seats
        if session.capacity > 0:
            session.occupied_percentage = (session.occupied_seats / session.capacity) * 100
        else:
            session.occupied_percentage = 0.0

def f_create_prueba(self):
    # [Rubrica: ORM Create]
    sesion = {
        "name": "Clase de Prueba Generada",
        "capacity": 20,
        "duration": 90
    }
    print("Creando sesión desde código:", sesion)
    self.env['gym.session'].create(sesion)

```

```

def f_search_update(self):
    # [Rubrica: ORM Search y Write]
    sesion = self.env['gym.session'].search([('capacity', '=', 10)], limit=1)
    print('Sesión encontrada:', sesion.name)
    sesion.write({
        "capacity": 15
    })

```

```

# 5. MODELO RESERVA
class GymBooking(models.Model):
    _name = 'gym.booking'
    _description = 'Reserva de Plaza'

    code = fields.Char(string="Código Reserva", readonly=True, default="BORRADOR")

    # [Rubrica: Lambda Default]
    booking_date = fields.Datetime(string="Fecha Reserva", default=lambda self: fields.Datetime.now())

    session_id = fields.Many2one("gym.session", string="Clase", required=True)
    member_id = fields.Many2one("res.partner", string="Socio", required=True, domain="|('is_gym_member', '=', True)]")

    state = fields.Selection([('draft', 'Pendiente'),
                             ('confirmed', 'Confirmada'),
                             ('cancelled', 'Cancelada')], string="Estado",
                            default='draft')

@api.model
def create(self, vals):
    if vals.get('code', 'BORRADOR') == 'BORRADOR':
        vals['code'] = 'RES-' + str(random.randint(1000, 9999))
    return super(GymBooking, self).create(vals)

@api.constrains('session_id')
def _check_capacity(self):
    for booking in self:
        if booking.session_id.available_seats < 0:
            raise ValidationError("¡Error! No quedan plazas libres en esta clase.")

def action_confirm(self):
    for booking in self:
        booking.state = 'confirmed'

```

- Views: Archivos XML con la definición de interfaces

```

addons > gymangel > views > gym_views.xml
 1  <odoo>
 2    <data>
 3
 4      <!--1. MENÚS (Navegación)-->
 5
 6      <menuitem id="menu_gym_root" name="Gimnasio Angel" web_icon="gym_angel,static/description/icon.png"/>
 7
 8      <menuitem id="menu_gym_management" name="Gestión" parent="menu_gym_root" sequence="10"/>
 9      <menuitem id="menu_gym_config" name="Configuración" parent="menu_gym_root" sequence="20"/>
10
11      <!--2. Vistas del modelo sesión-->
12      <!--vista kanban-->
13      <record id="view_gym_session_form" model="ir.ui.view">
14        <field name="name">gym.session.form</field>
15        <field name="model">gym.session</field>
16        <field name="arch" type="xml">
17          <form string="Sesión">
18            <sheet>
19              <div class="oe_title">
20                <h1><field name="name"/></h1>
21              </div>
22
23              <group>
24                <group>
25                  <field name="activity_id"/>
26                  <field name="instructor_id"/>
27                  <field name="occupied_seats"/>
28                  <field name="occupied_percentage" widget="progressbar" string="Progreso Ocupación"/>
29                </group>
30                <group>
31                  <field name="start_date"/>
32                  <field name="duration"/>
33                  <field name="capacity"/>
34                </group>
35              </group>
36
37              <notebook>
38                <page string="Asistentes">
39                  <field name="booking_ids">
40                    <tree editable="bottom">
41                      <field name="member_id"/>
42                      <field name="state"/>
43                    </tree>
44                  </field>
45                </page>
46              </notebook>
47            </sheet>
48          </form>
49        </field>
50      </record>
51
52      <record id="view_gym_session_tree" model="ir.ui.view">
53        <field name="name">gym.session.tree</field>
54        <field name="model">gym.session</field>
55        <field name="arch" type="xml">
56          <tree decoration-danger="available_seats == 0" decoration-info="available_seats &gt; 0">
57            <field name="name"/>
58            <field name="instructor_id"/>
59            <field name="start_date"/>
60            <field name="available_seats"/>
61          </tree>
62        </field>
63      </record>
64

```

```

<record id="view_gym_session_kanban" model="ir.ui.view">
    <field name="name">gym.session.kanban</field>
    <field name="model">gym.session</field>
    <field name="arch" type="xml">
        <kanban default_group_by="activity_id" class="o_kanban_small_column">
            <field name="name"/>
            <field name="instructor_id"/>
            <field name="available_seats"/>
            <field name="color"/>

            <templates>
                <t t-name="kanban-box">
                    <div t-attf-class="oe_kanban_color #{kanban_getcolor(record.color.raw_value)} oe_kanban_card oe_kanban_global_click">
                        <div class="o_dropdown_kanban dropdown">
                            <a class="dropdown-toggle o-no-caret btn" role="button" data-toggle="dropdown" href="#" aria-label="Dropdown menu" title="Dropdown menu">
                                <span class="fa fa-ellipsis-v"/>
                            </a>
                            <div class="dropdown-menu" role="menu">
                                <t t-if="widget.editable"><a role="menuitem" type="edit" class="dropdown-item">Editar</a></t>
                                <t t-if="widget.deletable"><a role="menuitem" type="delete" class="dropdown-item">Borrar</a></t>
                                <ul class="oe_kanban_colorpicker" data-field="color"/>
                            </div>
                        </div>
                    <div class="oe_kanban_content">
                        <div class="o_kanban_record_top">
                            <strong class="o_kanban_record_title">
                                <field name="name"/>
                            </strong>
                        </div>
                        <div class="o_kanban_record_body">
                            <span class="text-muted">Monitor: </span>
                            <field name="instructor_id"/>
                        </div>
                        <div class="o_kanban_record_bottom">
                            <div class="oe_kanban_bottom_left">
                                <span class="badge badge-primary">
                                    Libres: <field name="available_seats"/>
                                </span>
                            </div>
                        </div>
                    </div>
                </t>
            </templates>
        </kanban>
    </field>
</record>

```

```

<record id="action_gym_session" model="ir.actions.act_window">
    <field name="name">Clases Programadas</field>
    <field name="res_model">gym.session</field>
    <field name="view_mode">kanban,tree,form</field>
</record>

<menuitem id="menu_gym_session" parent="menu_gym_management" action="action_gym_session"/>

```

```

<!--3. Vistas del modelo reserva-->
<!--barra de estado (statusbar)-->
<record id="view_gym_booking_form" model="ir.ui.view">
    <field name="name">gym.booking.form</field>
    <field name="model">gym.booking</field>
    <field name="arch" type="xml">
        <form>
            <header>
                <button name="action_confirm"
                    string="Confirmar"
                    type="object"
                    class="oe_highlight"
                    invisible="state != 'draft'"/>
                <field name="state" widget="statusbar"/>
            </header>
            <sheet>
                <div class="oe_title">
                    <h1><field name="code"/></h1>
                </div>
                <group>
                    <group>
                        <field name="member_id"/>
                        <field name="session_id"/>
                    </group>
                    <group>
                        <field name="booking_date"/>
                    </group>
                </group>
            </sheet>
        </form>
    </field>
</record>

<record id="view_gym_booking_tree" model="ir.ui.view">
    <field name="name">gym.booking.tree</field>
    <field name="model">gym.booking</field>
    <field name="arch" type="xml">
        <tree>
            <field name="code"/>
            <field name="session_id"/>
            <field name="member_id"/>
            <field name="state" widget="badge" decoration-success="state == 'confirmed'" decoration-muted="state == 'draft'"/>
        </tree>
    </field>
</record>

<record id="action_gym_booking" model="ir.actions.act_window">
    <field name="name">Reservas</field>
    <field name="res_model">gym.booking</field>
    <field name="view_mode">tree,form</field>
</record>

<menutem id="menu_gym_booking" parent="menu_gym_management" action="action_gym_booking"/>

```

```
<!--4. Vistas del modelo res.partner (socios y monitores)-->
<!--modificar vista existente con XPath-->
<record id="view_gym_member_inherit" model="ir.ui.view">
    <field name="name">res.partner.gym.inherit</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form"/>
    <field name="arch" type="xml">

        <xpath expr="//notebook/page[@name='internal_notes']" position="after">
            <page string="Gimnasio" name="gym_page">
                <group>
                    <field name="is_gym_member"/>
                    <field name="is_instructor"/>
                </group>
            </page>
        </xpath>
    </field>
</record>

<record id="action_gym_member" model="ir.actions.act_window">
    <field name="name">Socios</field>
    <field name="res_model">res.partner</field>
    <field name="view_mode">tree,form</field>
    <field name="domain">[('is_gym_member', '=', True)]</field>
    <field name="context">{'default_is_gym_member': True}</field>
</record>

<menuitem id="menu_gym_member" parent="menu_gym_management" action="action_gym_member"/>
```

```

<!--5. Vistas del modelo actividad, equipamiento e instructor-->
<record id="action_gym_activity" model="ir.actions.act_window">
    <field name="name">Actividades</field>
    <field name="res_model">gym.activity</field>
    <field name="view_mode">tree,form</field>
</record>

<record id="action_gym_equipment" model="ir.actions.act_window">
    <field name="name">Equipamiento</field>
    <field name="res_model">gym.equipment</field>
    <field name="view_mode">tree,form</field>
</record>

<menuitem id="menu_gym_activity" parent="menu_gym_config" action="action_gym_activity"/>
<menuitem id="menu_gym_equipment" parent="menu_gym_config" action="action_gym_equipment"/>

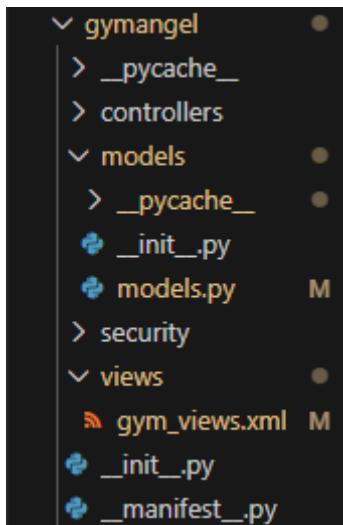
<record id="action_gym_instructor" model="ir.actions.act_window">
    <field name="name">Monitores</field>
    <field name="res_model">res.partner</field>
    <field name="view_mode">tree,form</field>
    <field name="domain">[('is_instructor', '=', True)]</field>
    <field name="context">{'default_is_instructor': True}</field>
</record>

<menuitem id="menu_gym_instructor"
          name="Monitores"
          parent="menu_gym_config"
          action="action_gym_instructor"
          sequence="5"/>

</data>
</odoo>

```

- Controllers. Endpoint para la API REST.



```

addons > gymangel > controllers > controllers.py > GymController > get_sessions
1  # -*- coding: utf-8 -*-
2  from odoo import http
3  from odoo.http import request
4  import json
5
6  class GymController(http.Controller):
7
8      # 1. Endpoint para obtener la lista de clases (GET)
9      # Se accede por: http://localhost:8069/api/gym/sessions
10     @http.route('/api/gym/sessions', auth='public', methods=['GET'], csrf=False)
11     def get_sessions(self, **kw):
12         # Buscamos todas las sesiones en la base de datos
13         sessions = request.env['gym.session'].sudo().search([])
14
15         # Preparamos una lista de diccionarios (JSON compatible)
16         data = []
17         for s in sessions:
18             data.append({
19                 'id': s.id,
20                 'nombre': s.name,
21                 'actividad': s.activity_id.name,
22                 'fecha': str(s.start_date),
23                 'plazas_libres': s.available_seats,
24                 'porcentaje_ocupacion': s.occupied_percentage,
25             })
26
27         # Devolvemos la respuesta en formato JSON
28         return request.make_response(
29             json.dumps(data),
30             headers=[('Content-Type', 'application/json')])
31

```

Lógica de negocio (Python): Se ha implementado una restricción crítica para el negocio: el control de aforo.

```

@api.constrains('session_id')
def _check_capacity(self):
    for booking in self:
        if booking.session_id.available_seats < 0:
            raise ValidationError("¡Error! No quedan plazas libres en esta clase.")

```

Este código valida que available_seats nunca sea menos que cero

Se ha diseñado una vista Kanban para facilitar la gestión visual de clases

```

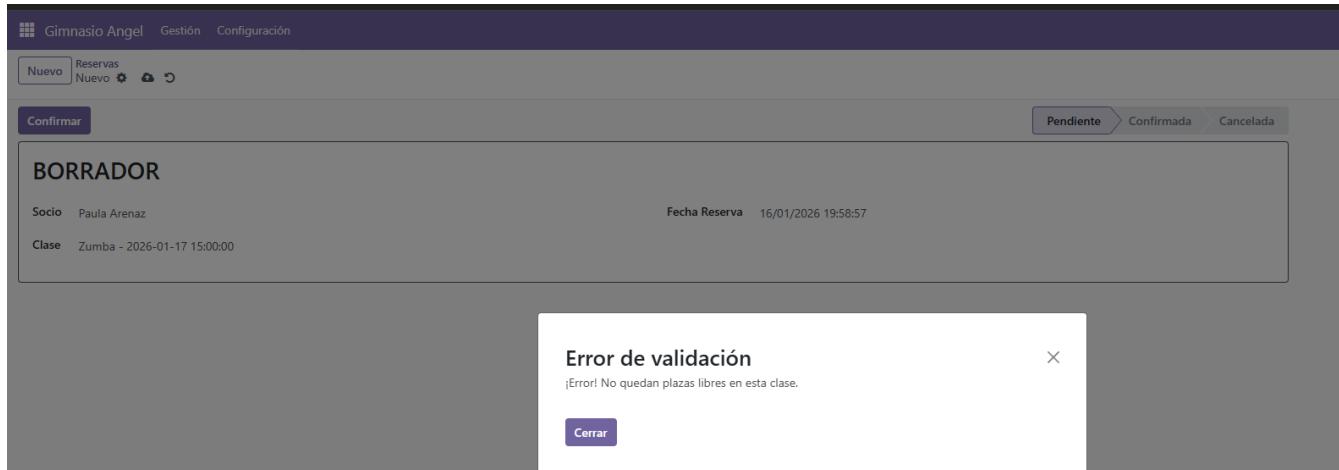
<record id="view_gym_session_kanban" model="ir.ui.view">
    <field name="name">gym.session.kanban</field>
    <field name="model">gym.session</field>
    <field name="arch" type="xml">
        <kanban default_group_by="activity_id" class="o_kanban_small_column">
            <field name="name"/>
            <field name="instructor_id"/>
            <field name="available_seats"/>
            <field name="color"/>

            <t t-name="kanban-box">
                <div t-attf-class="oe_kanban_color_#{kanban_getcolor(record.color.raw_value)} oe_kanban_card oe_kanban_global_click">
                    <div class="o_dropdown_kanban dropdown">
                        <a class="dropdown-toggle o-no-caret btn" role="button" data-toggle="dropdown" href="#" aria-label="Dropdown menu" title="Dropdown menu">
                            <span class="fa fa-ellipsis-v"/>
                        </a>
                        <div class="dropdown-menu" role="menu">
                            <t t-if="widget.editable"><a role="menuitem" type="edit" class="dropdown-item">Editar</a></t>
                            <t t-if="widget.deletable"><a role="menuitem" type="delete" class="dropdown-item">Borrar</a></t>
                            <ul class="oe_kanban_colorpicker" data-field="color"/>
                        </div>
                    </div>
                <div class="oe_kanban_content">
                    <div class="o_kanban_record_top">
                        <strong class="o_kanban_record_title">
                            <field name="name"/>
                        </strong>
                    </div>
                    <div class="o_kanban_record_body">
                        <span class="text-muted">Monitor: </span>
                        <field name="instructor_id"/>
                    </div>
                    <div class="o_kanban_record_bottom">
                        <div class="oe_kanban_bottom_left">
                            <span class="badge badge-primary">
                                Libres: <field name="available_seats"/>
                            </span>
                        </div>
                    </div>
                </div>
            </div>
        </t>
    </kanban>
</record>

```

4.4. Pruebas de funcionamiento

Prueba 1: Validación de Aforo (manejo de errores): Se intenta realizar una reserva en una clase que ya tiene 0 plazas libres:



The screenshot shows a web application for Gimnasio Angel. At the top, there's a navigation bar with 'Gimnasio Angel', 'Gestión', and 'Configuración'. Below it, a sub-menu has 'Nuevo' selected, followed by 'Clases Programadas' and 'Zumba - 2026-01-17 15:00:00' with a gear icon.

Zumba - 2026-01-17 15:00:00

Actividad	Zumba	Fecha Inicio	17/01/2026 16:00:00														
Instructor	Pablo Dominguez	Duración (min) ?	60														
Ocupadas	5	Capacidad Máxima	5														
Progreso Ocupación	<div style="width: 100%;">100 %</div>																
<input type="button" value="Asistentes"/> <table border="1"> <thead> <tr> <th>Socio</th> <th>Estado</th> </tr> </thead> <tbody> <tr> <td>Daniel Gutierrez</td> <td>Confirmada</td> </tr> <tr> <td>David Hernandez</td> <td>Pendiente</td> </tr> <tr> <td>Julia Pérez</td> <td>Confirmada</td> </tr> <tr> <td>Lucía García</td> <td>Confirmada</td> </tr> <tr> <td>Ángel Maroto</td> <td>Confirmada</td> </tr> <tr> <td colspan="2">Añadir una línea</td> </tr> </tbody> </table>				Socio	Estado	Daniel Gutierrez	Confirmada	David Hernandez	Pendiente	Julia Pérez	Confirmada	Lucía García	Confirmada	Ángel Maroto	Confirmada	Añadir una línea	
Socio	Estado																
Daniel Gutierrez	Confirmada																
David Hernandez	Pendiente																
Julia Pérez	Confirmada																
Lucía García	Confirmada																
Ángel Maroto	Confirmada																
Añadir una línea																	

Prueba 2: Consulta API REST, se realiza una petición HTTP GET al endpoint y el navegador devuelve un JSON con la lista de clases activas:

Se accede a la petición desde esta dirección:

<http://localhost:8069/api/gym/sessions>

The screenshot shows a Google Chrome browser window with the URL 'localhost:8069/api/gym/sessions' in the address bar. The page content displays a JSON array:

```
[{"id": 4, "nombre": "Zumba - 2026-01-17 15:00:00", "actividad": "Zumba", "fecha": "2026-01-17 15:00:00", "plazas_libres": 0, "porcentaje_ocupacion": 100.0}]
```

```

# -*- coding: utf-8 -*-
from odoo import http
from odoo.http import request
import json

class GymController(http.Controller):

    # 1. Endpoint para obtener la lista de clases (GET)
    # Se accede por: http://localhost:8069/api/gym/sessions
    @http.route('/api/gym/sessions', auth='public', methods=['GET'], csrf=False)
    def get_sessions(self, **kw):
        # Buscamos todas las sesiones en la base de datos
        sessions = request.env['gym.session'].sudo().search([])

        # Preparamos una lista de diccionarios (JSON compatible)
        data = []
        for s in sessions:
            data.append({
                'id': s.id,
                'nombre': s.name,
                'actividad': s.activity_id.name,
                'fecha': str(s.start_date),
                'plazas_libres': s.available_seats,
                'porcentaje_ocupacion': s.occupied_percentage,
            })

        # Devolvemos la respuesta en formato JSON
        return request.make_response(
            json.dumps(data),
            headers=[('Content-Type', 'application/json')]
        )

```

Esta es la clase que nos permite gestionar la petición GET, esta clase lo que hace es recuperar los datos de las sesiones creadas y se devuelve el contenido conseguido en formato JSON.

4.5. Despliegue de la aplicación

Para este proyecto se ha optado por un despliegue en servidor local virtualizado, utilizando para ello Docker con Docker Compose. Esta arquitectura permite encapsular todas las dependencias del sistema, como son librerías de Python, garantizando que la aplicación funcione idénticamente en cualquier máquina anfitriona, independientemente del sistema operativo.

El sistema se compone de dos contenedores:

1. Servidor Web con Odoo, que contiene la lógica de negocio, el cliente web y la API REST.
2. Base de Datos con PostgreSQL, almacena de forma persistente la información de socios, clases y demás datos.

Ambos servicios se comunican dentro de una red interna virtual creada por Docker.

Se utiliza un fichero de configuración que es el docker-compose.yaml:

Aquí se detallan los parámetros clave para cargar el módulo.

```
docker-compose.yaml
1  version: "3.8"
2  services:
3
4    odoop:
5      image: odoo:17
6      container_name: odoop
7      restart: unless-stopped
8      links:
9        - db:db
10       depends_on:
11         - db
12       ports:
13         #puerto de acceso web y API.
14         - "8069:8069"
15       volumes:
16         - odoo-data:/var/lib/odoo
17         - ./config:/etc/odoo
18         #mapeo de los módulos en este caso 'gym_angel'
19         - ./addons:/mnt/extr-addons
20       networks:
21         - red_odoop
22
23    db:
24      image: postgres:latest
25      container_name: container-postgresdb
26      restart: unless-stopped
27      environment:
28        - DATABASE_HOST=127.0.0.1
29        - POSTGRES_DB=postgres
30        - POSTGRES_PASSWORD=odoo
31        - POSTGRES_USER=odoo
32        - PGDATA=/var/lib/postgresql/data/pgdata
33      volumes:
34        - db-data:/var/lib/postgresql/data
35      networks:
36        - red_odoop
37      ports:
38        - "5432:5432"
39
40
41    pgadmin:
42      image: dpage/pgadmin4:latest
43      depends_on:
44        - db
45      ports:
46        - "80:80"
47      environment:
48        PGADMIN_DEFAULT_EMAIL: pgadmin4@pgadmin.org
49        PGADMIN_DEFAULT_PASSWORD: admin
50      restart: unless-stopped
51      networks:
52        - red_odoop
53
54    volumes:
55      odoo-data:
56      db-data:
57
58    networks:
59      red_odoop:
```

La ejecución del Entorno se hace de la siguiente manera:

1. Arranque del sistema con docker-compose up -d
2. Reinicio tras cambios con docker-compose restart
3. Monitorización de logs con docker logs -f odoo

5. Manuales

5.1. Manual de usuario:

A. Gestión de Monitores y Socios:

Para diferenciar un cliente estándar es decir un socio, de un empleado del gimnasio, es decir, un monitor, el sistema añade nuevas opciones en la ficha de contactos heredado de res.partner.

1. Navegar al menú Configuración > Monitores

The screenshot shows two parts of the Odoo application. The top part is a configuration screen for 'Monitores' under the 'Configuración' tab. It has a sidebar with options: 'Monitores' (selected), 'Actividades', and 'Equipamiento'. The main area lists contacts with checkboxes for 'Nombre': 'Marta Caballero' and 'Pablo Lopez'. The bottom part shows a detailed view of a contact record for 'Antonio Lopez'. The contact form includes fields for 'Nombre', 'Contacto' (Address), 'Puesto de trabajo' (Job Title), 'Teléfono' (Phone), 'Móvil' (Mobile), 'Correo electrónico' (Email), 'Sitio web' (Website), 'Título' (Title), and 'Etiquetas' (Tags). At the bottom of the contact form, there are checkboxes for '¿Es Socio?' (Is Member?) and '¿Es Monitor?' (Is Monitor?). The '¿Es Monitor?' checkbox is checked and highlighted with a red border.

Podemos comprobar que cuando creamos un monitor desde esta pestaña está marcado de forma automática la casilla de Es Monitor, lo que crea un contacto como monitor, también tiene la posibilidad de ser socio.

Para crear un socio hay que navegar a Gestión > Socios:



Y comprobamos que creando un Socio desde esta ventana la casilla de ¿Es Socio? Aparece marcada por defecto.

This screenshot shows the 'Nuevo' (New) screen for creating a new member ('Socio'). The form includes fields for basic information like name ('Luis Díaz'), address ('Calle...'), phone ('Teléfono'), and email ('Correo electrónico'). It also includes sections for work ('Puesto de trabajo') and website ('Sitio web'). At the bottom, there are checkboxes for '¿Es Socio?' (checked) and '¿Es Monitor?' (unchecked). The 'Gimnasio' tab is selected at the bottom right.

B. Planificación de Sesiones (Clases)

El coordinador del gimnasio puede programar nuevas clases desde el menú Gestión > Clases Programadas:



1. Pulsar en Nuevo
2. Seleccionar Actividad (ej: Yoga, Zumba) y el instructor
3. Definir la fecha de inicio y la duración, la fecha de fin se calcula automáticamente.
4. Establecer la Capacidad Máxima (Aforo).
5. Visualización: La barra de progreso mostrara en tiempo real el porcentaje de ocupación (0% inicialmente).

The screenshot shows a software application window for 'Gimnasio Angel'. At the top, there's a navigation bar with icons for home, gestion, configuration, and user profile. Below the bar, a sub-menu shows 'Nuevo' selected, followed by 'Clases Programadas' and 'Spinning - 2026-01-19 11:22:59'. On the right, there are buttons for '1 / 1' and navigation arrows. The main content area is titled 'Spinning - 2026-01-19 11:22:59'. It displays session details: Actividad (Spinning), Instructor (Marta Caballero), Fecha Inicio (19/01/2026 12:22:59), Duración (min) (60), Ocupadas (4), and Capacidad Máxima (5). A progress bar indicates 'Progreso Ocupación' at 80%. Below this, a table lists 'Asistentes' (Participants) with columns for 'Socio' (Member) and 'Estado' (Status). The participants listed are Angel Maroto, David Hernandez, Paula Arenaz, and Sofia Dominguez, all marked as 'Confirmada' (Confirmed). There's also a link to 'Añadir una línea' (Add a line). On the left, there's a sidebar with a 'Asistentes' tab and a search bar containing 'Escriba algo...'.

C. Control de Reservas y Aforo.

Dentro de una sesión en la pestaña asistentes se pueden añadir socios que acudirán a la clase, para ellos clicamos en Añadir una línea y nos saldrá la lista de socios del gimnasio, cuando añadimos uno podemos cambiar el estado de la reserva a confirmada o cancelada, por defecto será pendiente.

This screenshot shows the 'Asistentes' (Participants) section of the software. It lists members with their current status: Angel Maroto (Confirmada), David Hernandez (Confirmada), Paula Arenaz (Pendiente), and Sofia Dominguez (Confirmada). Below this, a dropdown menu is open for a member named Yubo Lin, showing options: Pendiente (selected), Pendiente, Confirmada, and Cancelada. The interface includes a search bar with placeholder text 'Escriba algo...' and a 'Buscar más...' button.

- Calculo Automático: Al añadir los asistentes, los campos “Ocupadas” y “Libres” se actualizan automáticamente.
- Restricción de Aforo: El sistema impide confirmar una reserva si no quedan plazas libres (available_seats < 0), lanzando un aviso de error al usuario para evitar el overbooking.



D. Vista General (Table Kanban).

Para una gestión visual rápida, el menú principal muestra las clases en formato de tarjetas(Kanban), agrupadas por tipo de actividad. Cada tarjeta ofrece un resumen con:

- Nombre de la sesión
- Monitor asignado
- Indicador de plazas libres
- Barra de color lateral para categorización visual.

Gimnasio Angel Gestión Configuración

Nuevo Clases Programadas

Spinning +

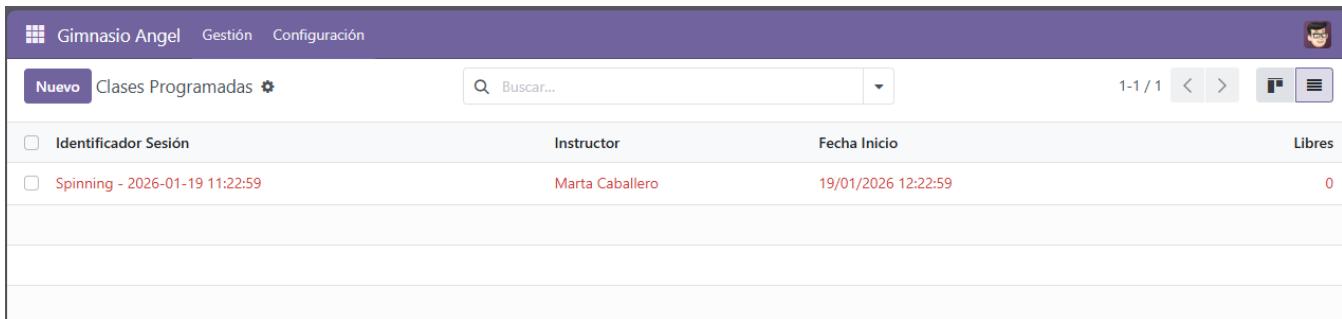
Spinning - 2026-01-19 11:22:59
Monitor: Marta Caballero
Libres: 0

Gimnasio Angel Gestión Configuración

Nuevo Reservas

Código Reserva	Clase	Socio	Estado
RES-1206	Spinning - 2026-01-19 11:22:59	Angel Maroto	Confirmada
RES-7422	Spinning - 2026-01-19 11:22:59	David Hernandez	Confirmada
RES-7995	Spinning - 2026-01-19 11:22:59	Paula Arenaz	Pendiente
RES-9913	Spinning - 2026-01-19 11:22:59	Sofia Dominguez	Confirmada
RES-9140	Spinning - 2026-01-19 11:22:59	Yubo Lin	Pendiente

Cuando una clase tiene 0 plazas libres la sesión aparece en rojo sino aparece en azul.



The screenshot shows a list of scheduled classes for 'Gimnasio Angel'. One class, 'Spinning - 2026-01-19 11:22:59', is listed with an availability status of '0' (red). The other class listed has 2 available slots (blue).

<input type="checkbox"/> Identificador Sesión	Instructor	Fecha Inicio	Libres
Spinning - 2026-01-19 11:22:59	Marta Caballero	19/01/2026 12:22:59	0



The screenshot shows a list of scheduled classes for 'Gimnasio Angel'. One class, 'Spinning - 2026-01-19 11:22:59', is listed with an availability status of '2' (blue).

<input type="checkbox"/> Identificador Sesión	Instructor	Fecha Inicio	Libres
Spinning - 2026-01-19 11:22:59	Marta Caballero	19/01/2026 12:22:59	2

5.2. Manual de instalación:

Requisitos previos a la instalación:

- Tener Instalado Docker y Docker Compose
- Disponer de una instancia de Odoo 17
- Acceso a la terminal del servidor

Paso 1: Despliegue del Código.

El módulo se distribuye como una carpeta llamada `gym_angel`. Esta carpeta debe copiarse dentro del directorio de addons montado en el contenedor de Odoo.

Paso 2: Reinicio de Servicios:

Tras añadir los archivos es necesario reiniciar el contenedor para que Python reconozca los nuevos archivos y controladores (API REST)

Utilizamos el comando `docker-compose restart`, si el contenedor no estaba activo primero necesitamos hacer `docker-compose up -d`.

Paso 3: Instalación en Odoo.

1. Acceder a Odoo con <http://localhost:8069> con credenciales de administrador
2. Activar el modo desarrollador (Ajustes > Activar modo desarrollador)
3. Ir al menú Aplicaciones
4. Pulsar en “Actualizar lista de aplicaciones” en la barra superior.
5. Buscar “Gimnasio Angel” en la barra de búsqueda.
6. Pulsar el botón **Activar**

Una vez instalado, aparecerá el nuevo menú raíz “Gimnasio Angel” y el sistema estará listo para usarse.

6. Conclusiones y posibles ampliaciones

6.1. Principales dificultades técnicas y soluciones:

Durante el ciclo de vida del desarrollo, se presentaron desafíos significativos que han servido para profundizar en el funcionamiento interno del ORM de Odoo y el lenguaje de Python:

1. El problema de la Dependencia Circular: El obstáculo más complejo surgió durante la instalación inicial del módulo. Se intentó aplicar un filtro de dominio (domain="["is_instructor", '=', True]"]") en una vista XML y en una acción de ventana, haciendo referencia a un campo (is_instructo) que se definía en el archivo Python en ese mismo momento.
 - a. El conflicto: Odoo intenta cargar y validar las vistas XML antes de confirmar la creación de todas las columnas en la base de datos PostgreSQL. Al no encontrar una columna is_instructor (porque la instalación aún no había finalizado), el sistema devolvía un Internal Server Error crítico, impidiendo la instalación.
 - b. La solución: Se adoptó una estrategia de Instalación en dos fases:
 - i. Se comentaron temporalmente en el código (XML y Python) todas las referencias al campo que nos estaba causando el problema.
 - ii. Se instaló el módulo “limpio” para forzar la creación de la estructura de base de datos.
 - iii. Se descomentó el código y se actualizó el módulo, permitiendo que las vistas validaran correctamente contra los campos ya existentes.

2. La sensibilidad de Python a la indentación: Al trabajar con múltiples archivos surgieron errores de indentación, Python utiliza la indentación para definir bloques lógicos, un solo espacio desalineado en el archivo models.py provocaba que el servidor de Odoo no pudiera arrancar.

6.2. Posibles ampliaciones:

1. Pasarela de pago: Integrar el módulo facturación para cobrar la cuota mensual a los socios automáticamente.
2. Envío de correos. Configurar plantillas de email para enviar una confirmación automática al realizar una reserva.

7. Bibliografía

Apuntes de clase:

Pdf: 2_DAM_SGE_UD5_Desarrollo_modulos.pdf

Pdf: 2_DAM_SGE_UD_6_Desarrollo_modulos_conceptos_avanzados.pdf

Documentación oficial de odoo:

Modelos y campos: Referencia sobre como crear modelos, campos computador y restricciones:

<https://www.odoo.com/documentation/17.0/developer/reference/backend/orm.html>

API REST: guía sobre el uso de http.route y json para crear la api del gimnasio:

<https://www.odoo.com/documentation/17.0/developer/reference/backend/http.html>

Gemini AI para resolver el problema de la dependencia circular y mejora de documentación.

Enlace a Github;

https://github.com/AngelMaroto/Practica11_SGE.git

Normas de entrega: **FECHA DE ENTREGA: 15 DE ENERO DE 2026**

- Es necesario incluir todos los apartados de la memoria. Extensión mínima: 20 páginas
- El proyecto tiene que estar subido a Github como público: El repositorio tiene que contener el código fuente de la aplicación y un archivo README con el siguiente contenido: Título del proyecto, descripción, enlace a la memoria del proyecto en formato PDF.
- Nombre del archivo de la memoria:
 - Apellido1_Apellido2_Nombre_Memoria_ProyectoOodoo_DAM25.pdf

Presentación oral del proyecto en clase: realizar una presentación de Powerpoint o guión para explicar la memoria. También es necesario demostrar que funciona la aplicación. Duración: 10 minutos